# IRIS BIOMETRIC USING ARTIFICIAL NEURAL NETWORKS

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical Engineering

by
Kevin Joseph Haskett
August 2018

COMMITTEE MEMBERSHIP

TITLE: Iris Biometric using Artificial Neural Networks

AUTHOR: Kevin Joseph Haskett

DATE SUBMITTED: August 2018

COMMITTEE CHAIR: Xiao-Hua (Helen) Yu, Ph.D.
Professor of Electrical Engineering

COMMITTEE MEMBER: Jane Zhang, Ph.D.
Professor of Electrical Engineering

COMMITTEE MEMBER: John A. Saghri, Ph.D.
Professor of Electrical Engineering

ABSTRACT

Iris Biometric using Artificial Neural Networks

Kevin Joseph Haskett

A biometric is a more secure way of personal identification than passwords. This thesis examines the iris as a personal identifier with use of a neural network as the classifier. A comparison of different feature extraction methods are performed, including the Fourier transform, discrete cosine transform, Eigen values, and the wavelet transform and the combinations of them. The robustness of each method, and different combinations of methods, with respect to distortion and noise, is also studied by adding noise applied to test images to the trained neural network.

Keywords: Neural Networks, Eigen Analysis, Discrete Cosine Transform, Wavelet Transform, Classifier, Iris Identification

# ACKNOWLEDGMENTS

I would first and foremost like to thank my thesis advisor Professor Xiao-Hua (Helen) Yu for guiding me throughput the long process of completing this thesis. Her continual guidance is greatly appreciated and kept me focused. I would also like to thank my parents for their patience and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# CHAPTER I: INTRODUCTION

A biometrics system provides automatic recognition of an individual based on a unique feature possessed by the individual and can play an important role in public and information security. For example, workers with access to a cash register can use a biometric instead of a password and increase security, by not having the chance to say a password to a coworker that could be overheard. Biometrics accurately identify individuals, whether it is through their fingerprint, hand geometry, voice, iris, or other unique characteristic [10].

Iris recognition biometrics is a popular means of identification, in the Parisian penal system when officials visually inspected inmate's irises [14]. Automation of iris identification is now a focus, and requirement, of using the iris as a biometric.

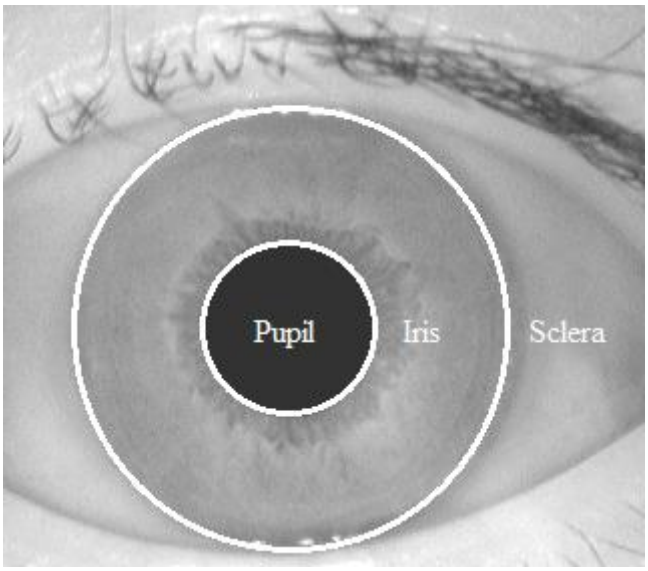



**Figure 1: Portions of an eye [2]**

The iris, colored region of eye, is a good attribute to base a biometric on because irises are extremely distinctive (even between an individual's left and right eye). Although coloring and a few other characteristics of the eye are hereditary the minute textures and patterns are based on circumstances one undergoes while the eye is being

developed. It is therefore extremely unlikely that the development process will be replicated. Also, after childhood, the iris changes very little, except for the slight depigmentation and shrinking of the pupil [14]. It is because of its uniqueness and stability that the iris is a reliable personal biometric [7].

Recognition of the iris consists of three steps. The first step is capturing the iris, which is done with a special close up iris camera. The second step is preprocessing and recognizing the iris, which includes detection and isolation. The iris is isolated by detecting its inner and outer boundaries. Since both the iris-pupil and iris-sclera boundary are comparable to perfect circles, a circular Hough transform is typically used. The third step is the identification step, where an image of an iris is compared to a database to find a match. Identification is where the most variability between algorithms occurs. Feature extraction, which is used for identification in step 3, can be focused on phase, wavelet transforms, Gabor filtering, or texture.

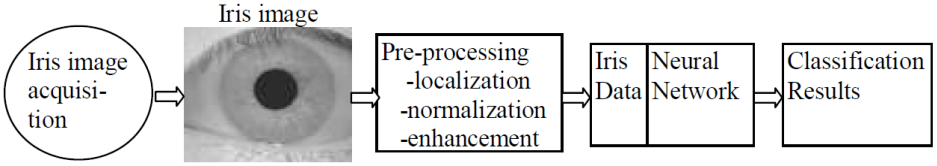



**Figure 2: The architecture of the iris recognition system [1]**

The approach taken in this thesis follows the three steps of iris recognition, with a few alterations for increased speed, robustness, and accuracy. Iris images are taken from the CASIA iris database [5]. Iris detection is done with a darkest region technique to detect the pupil-iris boundary and a circular Hough transform is used to detect the iris-sclera boundary. Finally, patterns within the iris is the feature used for identification and is captured with a variety of methods including the Fourier transform, the discrete cosine

transform, Eigen coefficients, and the wavelet transform. Analysis is done with each of these methods individually and various combinations, to find the most reliable and robust method.

There are a variety of methods to decrease the number of values needed to represent an image. The Fourier transform and the discrete cosine transform concentrate on the information of the different frequency components of the iris. The Eigen methodology focuses on spatial information and gives weights to the iris images in a predefined set that when linearly combined sum up to the iris image under test. Finally, the wavelet transform combines spatial and frequency components in its weights [17]. It is important to limit the amount of information needed to describe an image to decrease the time required to differentiate images.

This thesis is organized to follow the required steps in completing the work of iris identification. Chapter I studies the steps needed to go from an image of an eye to form inputs to a neural network. Chapters II through V studies the 4 different methods of feature extraction chosen for this work. Chapters VI though IX discuss results from the combination of feature extraction methods. Chapter X summarizes the results and draws conclusions of preferred feature extraction methods compared to others that are performed. And Chapter X1 discusses possible future work that could be done to optimize and enhance some of the methods studied in this thesis.

LITERATURE REVIEW

The first use of iris recognition as a personal identifier dates back over a hundred years in Paris when inmates of the penal system were identified by the color patterns of their iris. It wasn't until more recently when Flom and Safir in the early 1990s, proposed the concept of an automated iris recognition system, like the one researched here. Although these two never developed a working model, their idea has stemmed many studies into the field of automatic iris recognition [7].

One of the first and best iris recognition systems is proposed by Daugman in 1991 [10]. Daugman examines the uniqueness of an iris and determined that there are over 100 independent degrees of freedom, meaning, since there are so many properties that can change, the iris is an excellent determining factor in personal identification. The method used in his paper is to extract texture information from the iris, by applying a two dimensional Gabor filter to the iris region. The spatial filter (Gabor filter) is shown in figure 3 [10]. First, in Daugman's algorithm, the iris is detected by means of a slightly altered Hough transform that looks for a circle that has the maximum change in pixel values, and is then converted into a rectangle, using polar conversions. The 2D Gabor filter is then applied and its coefficients are then classified using the Hamming distance (weight of similarity of two iris coefficient sets). It is

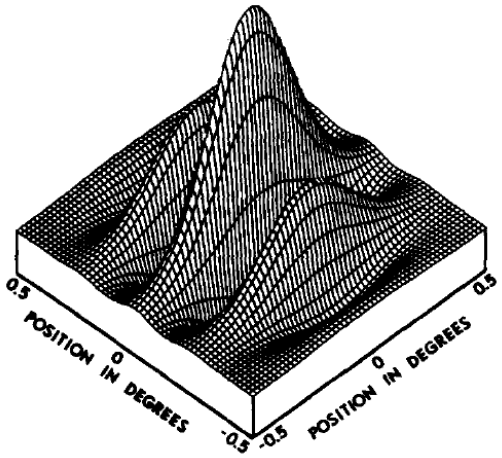



**Figure 3: Gabor spatial filter profile [10]**

reported that this method gives classification of 100 percent. The 100% accuracy is impressive, but tests to examine the robustness of the algorithm were not performed.

Another successful method of iris recognition is proposed by Richard P. Wildes [15]. Wildes' method is similar to that of Daugman's but instead of a 2D Gabor filter a Laplacian of Gaussian filter is convolved with the iris part of the image. The Laplacian of Gaussian kernel acts as a bandpass filter of the image (when convolved). The Wildes algorithm also treats the iris eyelid boundary as two different parabolas whereas the Daugman method ignores the top and bottom portions of the eye image. Based on one test that included 60 different iris images, accuracy of 100% is achieved.

A more recent method of iris recognition, introduced by Miyazawa [12], takes advantage of a two-dimensional Fourier transform to decrease the number of coefficients needed to represent the image (from millions to thousands). After the iris region has been located, the top half of the iris is unfolded into a rectangle, and then has its Fourier transform taken, shown in figure 4. As can be seen in the image of figure 4 the only coefficients needed to represent the iris region are the white pixels in the Fourier image. The classifier that Miyazawa's method uses [12] is a region matching algorithm shown in figure 5. Six different regions are compared against each other and if the correlation

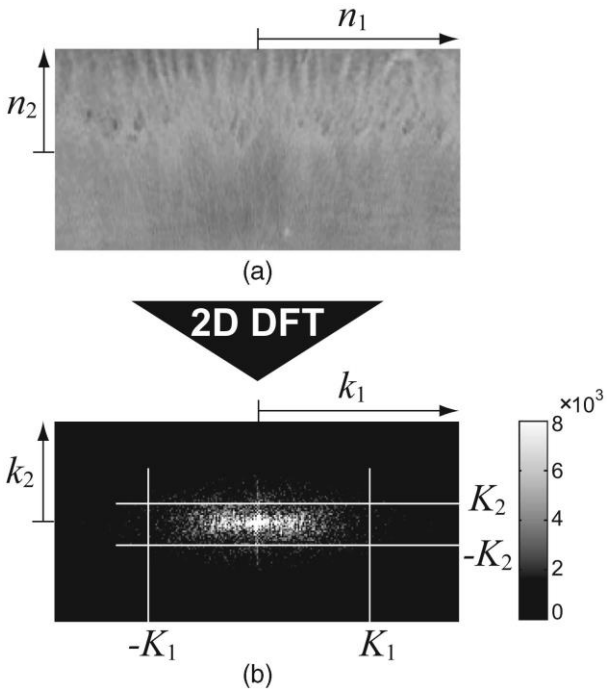



**Figure 4: Fourier transform of the upper half of a detected iris**

between the same regions of the different eyes is high (>0.6) then the two eyes are said to be a match.

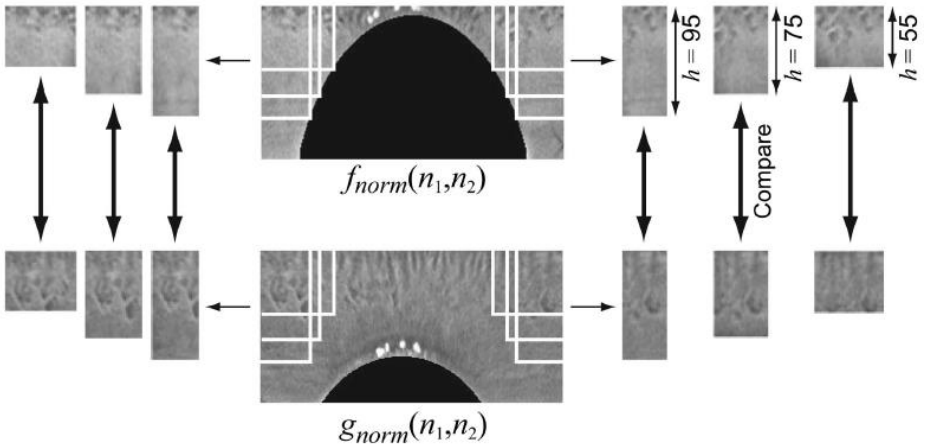



**Figure 5: Different regions for comparison in the matching algorithm**

Miyazawa's method, using the Fourier transform and its matching algorithm of six different regions of the Fourier transform coefficients achieves 93.2% correct classification using the CASIA iris image database [12].

The three methods discussed above all use a method to reduce the number of features needed to represent an image, but they do not use a neural network as their classifier. Abdel [6] uses the discrete cosine transform to reduce the number of image coefficients to 50, and then uses a neural network with one hidden layer as the classifier. Abdel also analyzes Daugman's [13] method (50 2D Gabor filter coefficients) but uses a neural network with two hidden layers as a classifier. Adbel's experiment results show 83.65% correct classification of phase coefficients and 96.1% correct classification when using 50 2D discrete cosine transform coefficients. Although these test results are promising for the use of a neural network, only three different eyes are used when testing the method. Abhiram [9] uses the discrete cosine transform but instead of using a neural network as a classifier, a binary particle swarm optimization algorithm is used as a

classifier and obtains 73.28%, 68.96%, and 94.27% correct classification for the LEI, CVonline, and CASIA eye image databases respectively. With Adbel and Abhirams experiments it can be seen that the neural network is as good as, or better than other classifiers when it comes to correctly classifying iris images.

EYE IMAGES

With the use of the iris as a personal biometric, it is necessary to test the identification methodology with a large variety of images. The acquisition of many iris images has already been performed by Biometric Ideal Test. The Chinese Academy of Sciences' Institute of Automation (CASIA) database includes seven images each, of 100 different people [5]. The size of the database allows for a huge variety of iris images. For example, in some of the images the iris is entirely visible and in others it is highly obstructed by the eyelid and/or the eyelash as shown in figure 6.

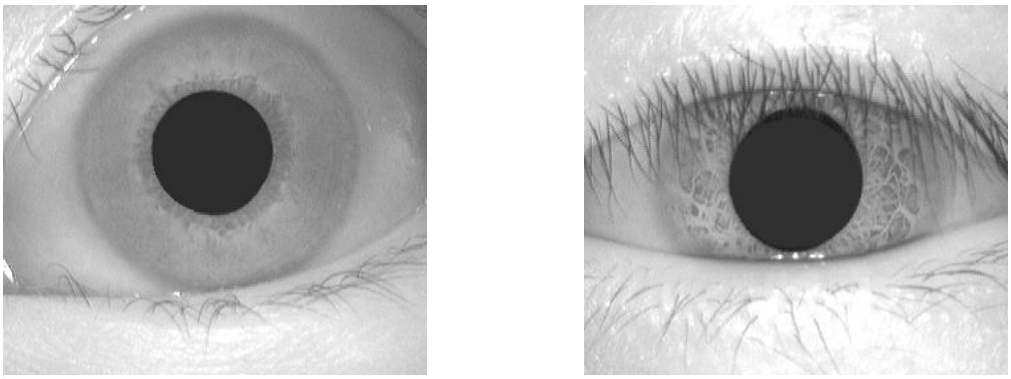


**Figure 6: (a) image with fully exposed iris (b) partially blocked iris [5]**

Other variations included in this database are the different locations of the pupil and iris. With a large variation of iris location and pupil shape, various obstructions from the eyelid, detection requires a robust algorithm for personal identification. Examples of images that are more difficult to distinguish are shown in figure 7 [5].

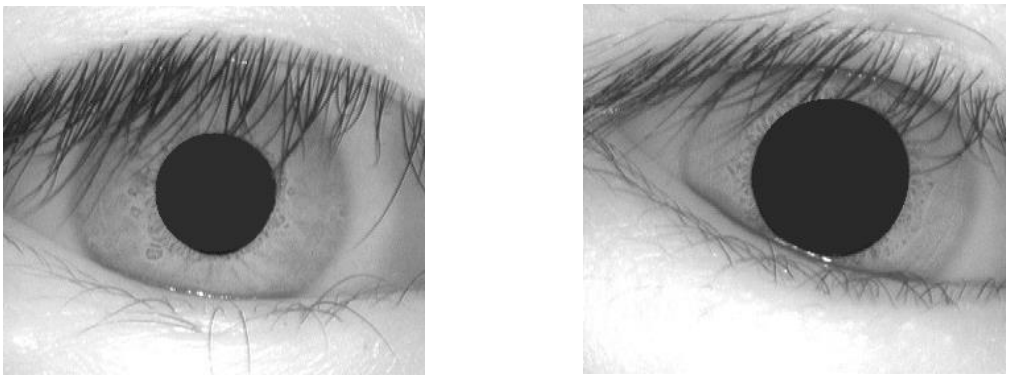


**Figure 7: Image variations include pupil size, location, and amount of eyelash obstruction**

There are both advantages and disadvantages with the use of the CASIA iris database. These iris images have good resolution with image sizes of 280x320 pixels of just the eye region. This is a huge advantage because it offers large amounts of detail in the iris region, making each individual eye easily distinguishable. A possible setback with such a high resolution image is that the method developed will only work for close up images of an iris, but not the one taken from distance. This narrows the scope of the recognition technique. Also, the specific database in use for the developed method has the pupil blacked out. There are many different techniques and algorithms that focus on recognizing the pupil in an image, and I decided to take advantage of the CASIA database that has this preprocessed.

PUPIL SEGMENTATION

Iris segmentation algorithms isolate the iris from the rest of the image. The first step in iris segmentation is locating the pupil by detecting the boundary between the pupil and the iris, which is done with a darkest square implementation [1]. To perform the darkest square implementation, it must be assumed that the pupil is the darkest portion of the entire eye image. The theory in implementing the darkest square algorithm is that, if you know the location of a pixel within the pupil, you can easily determine four pixels located on the boundary between the pupil and the iris. With the four boundary locations, both the center and radius of the pupil can be determined [1].

To implement the iris-pupil boundary location algorithm, we first separate the eye image into blocks of 10 X 10 pixels. A 10 X 10 block size is chosen [1] because it will eliminate random dark pixels that occur in eyelashes and eye corners from being determined as part of the pupil. It also allows multiple blocks to include part of the pupil. It is okay for edge pixels to be excluded from the blocks because the iris is not located in the edge pixels of the eye image. Next, find the three darkest blocks in the image and locate a pixel in the block that has all dark pixels in its four neighbor locations. The four neighbors are shown in figure 8, and are required to be dark to be certain that the pixel under consideration is not on the boundary of the pupil and the iris. Now with the three darkest pixels in each of the 3 darkest blocks, four pixels on the pupil –iris boundary can be found.

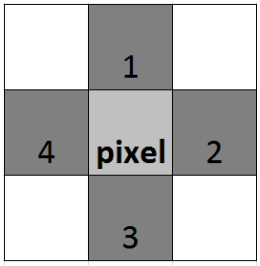



**Figure 8: Pixel and its four neighbors**

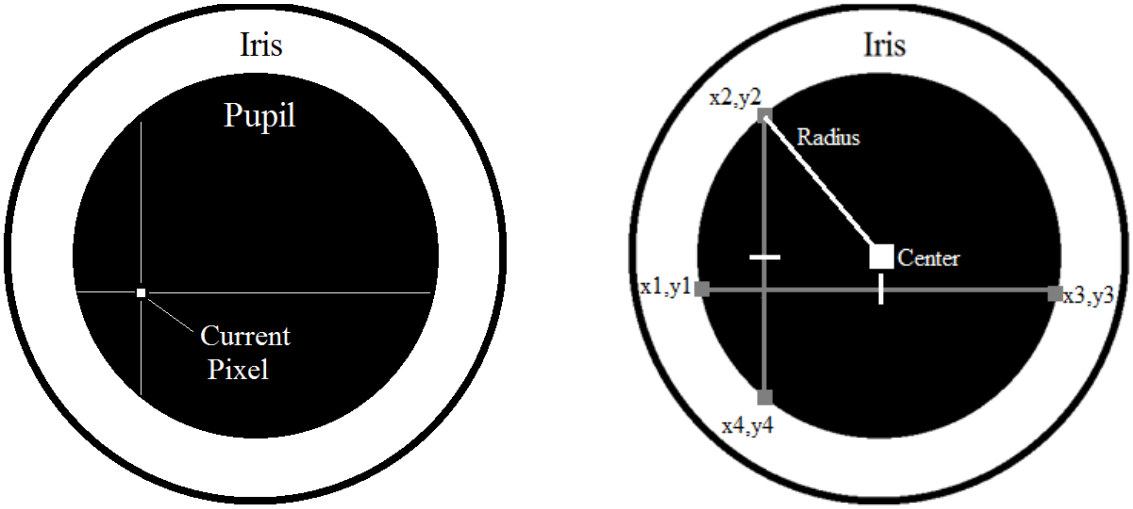



**Figure 9: (a) locating the 4 boundary pixels (b) finding the center and radius**

Starting at a location that is inside the pupil, extend in all four directions and look for the pupil-iris boundary. The boundary is the largest increase in pixel value (the dark pupil, low pixel intensity versus the lighter iris with higher pixel intensity). The center is now the average location in both directions and the radius is the average distance from the center pixel to the boundary pixels [1].

$$\textbf{center} = (x_c, y_c) = \left(\frac{1}{2} * (x_2 + x_4), \frac{1}{2} * (y_1 + y_3)\right) \quad (1)$$

$$\textbf{radius} = \frac{1}{4}\sum_{i=1}^{4}\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \quad (2)$$

The radius and the center of the pupil are found for the three darkest pixels, inside the pupil. To obtain a more accurate measurement of the radius and center, the three values are averaged to obtain a final radius and center. The images in figure 10 show the three initial estimations of the pupil (before they are averaged). It can be seen that the three initial pupil estimations may not be perfect (figure 10b), which is why an average of three pupil estimations is necessary.

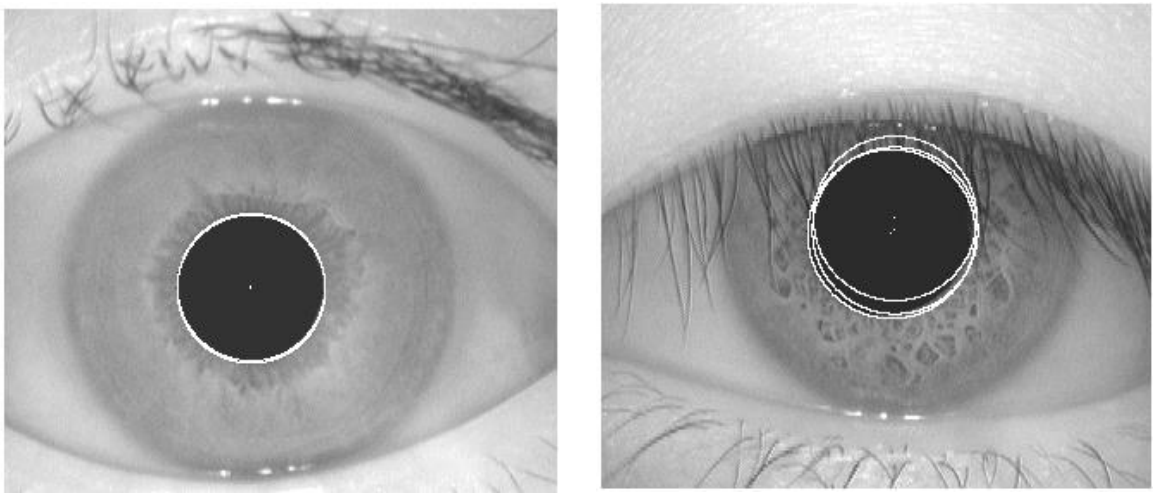


**Figure 10: Eye image with pupil and center locations (a) good detection (b) bad detection**

A pupil detection is considered passing if all three circles have a center within 2 pixels and radius within 2 pixels. The pupil detection algorithm was applied to 700 images from the CASIA database and had incorrect pupil labeling in 3 of the images. This error rate gives us a working pupil detection rate of 99.57%. An example of incorrect pupil detection is shown in figure 10b. Although the pupil is not exactly detected, the boundary that is found can still be used for further iris segmentation once the three estimations are averaged. (The pupil detection algorithm discussed, when implemented, takes 0.19 seconds per eye on a Lenovo Ideapad U410 with Intel i7-3537U CPU @ 2.00GHz).

IRIS SEGMENTATION

Once the boundary between the pupil and the iris is detected, the next step needed to isolate the iris is to find the boundary between the iris and the sclera (white part of eye). This boundary is harder to extract because the intensity difference between the iris and sclera can be small. To find this boundary, a circular Hough transform is performed, [14].

**Figure 11: Eye image (a) with edge detection (b) only possible iris boundary edges**

To increase the speed of the circular Hough transform, the number of edges must be decreased. To start, the edges of the eye image must be detected, which is done using a canny edge detector with a 13x13 kernel, to only catch the large edges. Edges near the outer boundary of the image are discarded as the iris will be near the center of entire image. Also, pixels too close to the pupil (within 40 pixels), pixels too far (not within 80 pixels) from the pupil, edges that are horizontal, or edges that are not part of a larger edge are all disregarded. Once these edges are eliminated from the eye's edge map, the resulting image is shown in figure 11b. The circular Hough transform is applied to the iris edge image shown in figure 11b.

The iris boundary detection algorithm was applied to 700 images from the CASIA database and, based upon an eye test, had correct detection for all images. An example of iris detection is shown in figure 12. It takes approximately 3.5 seconds per eye (Lenovo Ideapad U410 with Intel i7-3537U CPU at 2.00GHz) to run this algorithm, see appendix A for function "Locate Iris". An assumption that was made when locating the iris was that it will be a perfect circle and that the center of the iris is the same as the center of the pupil. Both are reasonable and the results are all positive.

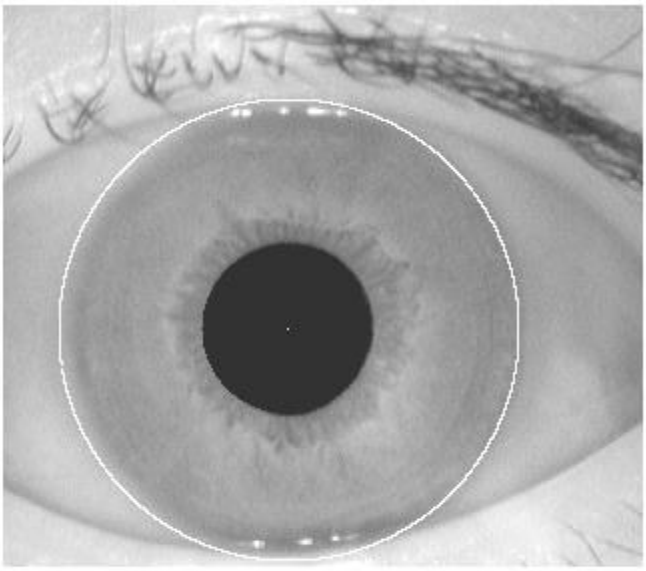


**Figure 12: Detection of the iris-sclera boundary**

The next step is to create a mask for each eye. This is done by multiplying all pixels that are outside the iris (inside the pupil or outside the iris-sclera boundary) by zero and all other pixels by one. This will leave an image where just the iris is visible, as shown in figure 13a. Since there may be eyelids and eyelashes in the masked image, shown in figure 13b, further improvement of the algorithm to eliminate the eyelashes and eyelids from the iris region could be implemented [10]. The biggest problem with this algorithm is the time it takes to run the circular Hough transform (appendix A, "Circle Hough"), approximately 3.5 seconds per eye. The speed could be increased by knowing facts that are observed from running the circular Hough transform on the eye images, including average, maximum, and minimum radius of the iris. This will limit the number of pixels the circular Hough transform looks at in determining the boundary between the sclera and iris.

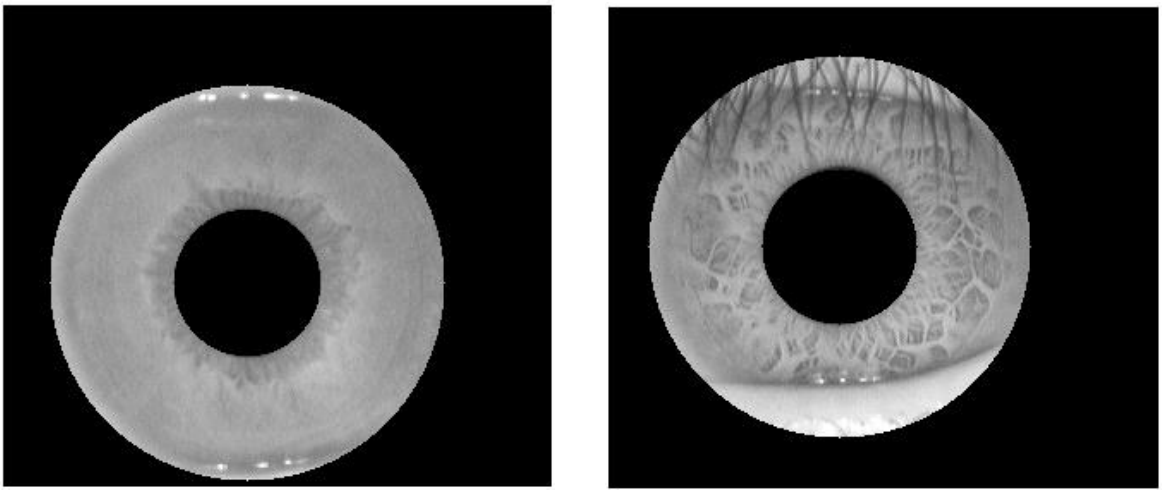


**Figure 13*:* Isolated iris (a) one without lashes inside iris region (b) lashes in the iris region**

When the Hough transform is run on the 700 eye images it is found that the largest iris radius is 120 pixels, the minimum is 89 pixels, and the average is 104.9 pixels. This information is used as inputs into the Hough algorithm, to limit the radius of the circle it is trying to detect. By using this range for the circular Hough transform it decreases the time per eye from 3.5 seconds to 2.2 seconds. Another improvement that can be made is to validate the radius of the pupil using the Circular Hough transform. This can be done since the both the iris boundaries can be approximated as circles. Although this would likely increase validity of the pupil-iris boundary it would come at a cost of time. This experiment was not perform in this research but could be performed for future analysis and improvements.

## OBTAINING INPUTS FROM THE EYES

Now that the process of obtaining the iris from an image has been selected the process in which the images are to be classified needs to be determined. A Neural network is used as the classifier of irises. Before a neural network can be used to classify images, it must first be trained with image coefficients, the fewer the better. Obtaining these coefficients will be discussed in detail later in the thesis. To train a network, images must be separated into training and testing images [9]. For this thesis, all training, test, and test images are taken from the CASIA iris database. The training images are the known inputs into the neural network to get the network coefficients to converge. For training data, I used 5 iris images from each person, depicted in figure 14a where each row contains images from the same person. One parameter that needs to be set before training the network is the desired mean squared error. This value represents how precise the outputs of the network match the desired classifications. The smaller the value, the more precise the network but the longer it takes for the neural network to converge. The desired mean square error for training in this thesis is less than 0.01. Once the neural network has converged to an appropriate mean squared error, the resultant weights are used to classify the test images. The images in figure 14b are the test images used on the neural network after training has been completed. These test images are of the same eyes as the training images but are different photos, as shown in figure 14. The differences between images of the same eye can be lighting, angle, eye offset, and the amount the iris is exposed.

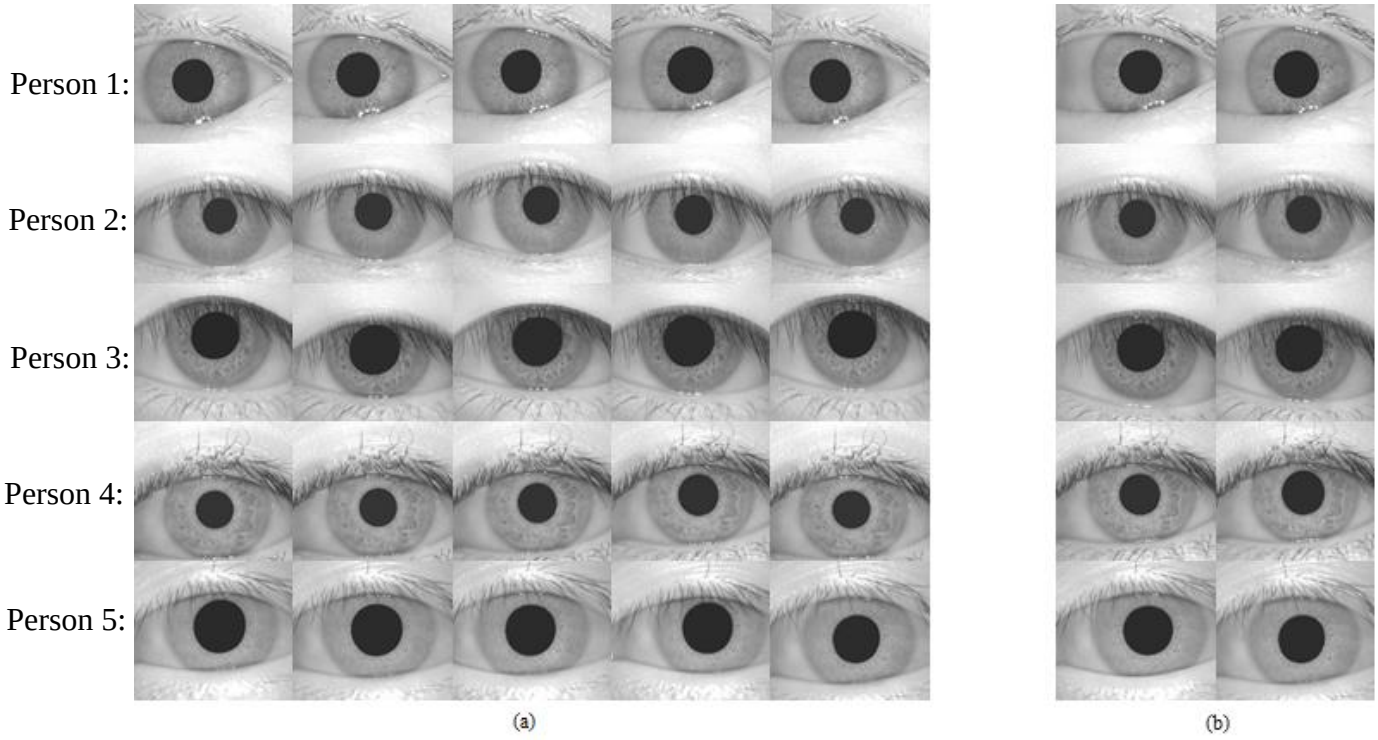



**Figure 14: (a) Training images (b) testing images**

TEST SETUP OPTIMIZATION

There are several things that need to be set up and tweaked before a test can be performed. First, the number of inputs to the neural network must be chosen. The number of inputs is the number of coefficients that each methodology (Fourier transform, discrete cosine transform, etc.) uses to describe the iris. The advantage of more coefficients is each eye will have more features to be distinguished from, but the disadvantage is that, with more coefficients, the neural network will take longer to train. Each methodology will have a different number of required coefficients.

The most significant contribution to training time is the size of the neural network [1]. The number of layers and the number of neurons in each layer both increase training time, but the network must be large enough to allow the error to converge. To find the optimum size of a neural network, analysis was performed on a network with 121 inputs and results can be seen in figure 15 (details in Appendix B). For all neural networks the images in figure 14a are the training images and the images in 14b are the test images.

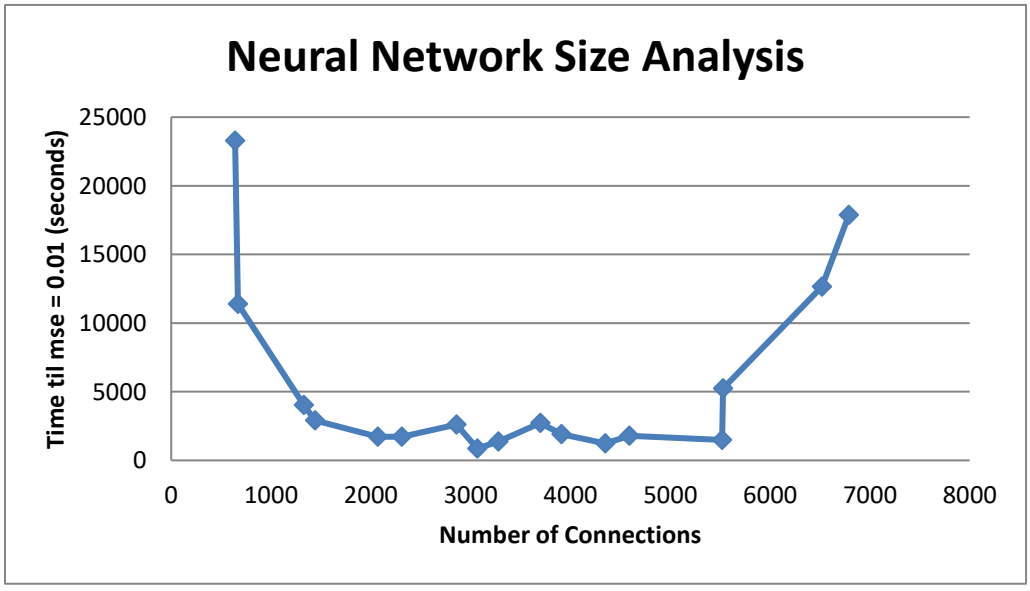



**Figure 15: Neural Network Size Analysis**

The number of coefficients are 121 because it was determined through trial and error that this number of inputs led to the fastest time of convergence for Fourier Transform coefficients, with a neural network of size 15, 21, 11, figure 16.

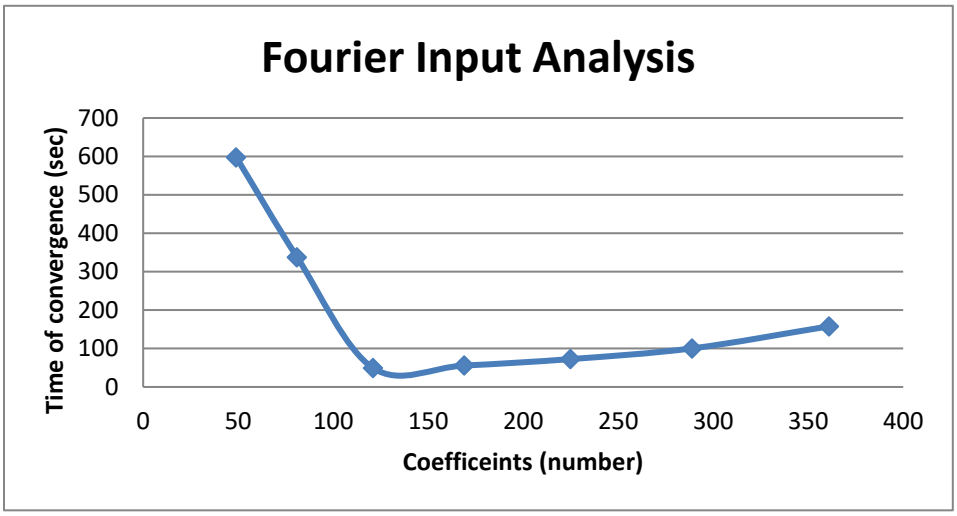



**Figure 16: Number of Fourier coefficients required to differentiate images**

If the neural network is size 2, 2, 3 with 121 inputs. The first hidden layer would have 2 neurons, the second hidden layer would have 2 neurons, and the third and final layer has 3 outputs. This would lead to (121*2) + (2*2) + (2*3) + (2+2+3) = 159 connections. Based on the analysis run with 17 different neural network sizes, see Appendix B for data, ranging from 1500 connections to 4500 connections, figure 16, the optimum network is a network of 15, 21, 11 with 3 outputs, giving 3068 connections. The plot has a 'U' shape. With too few connections the neural network will not converge quickly, and with too many connections it takes longer to train the network.

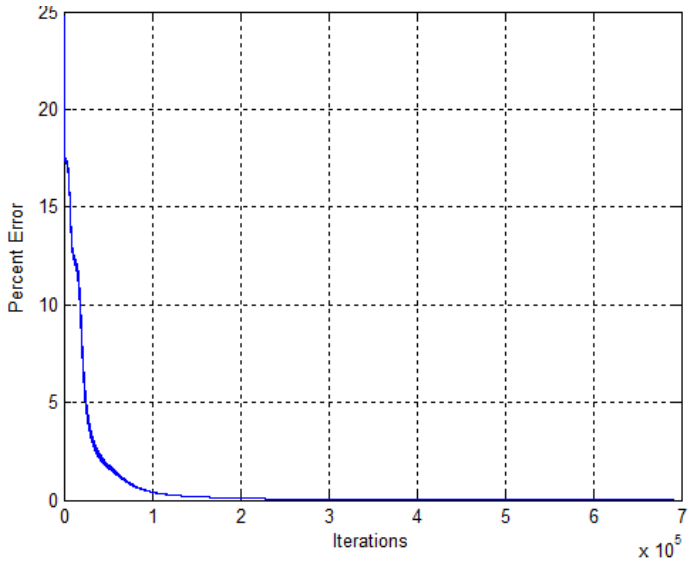



**Figure 17: Mean-square error convergence as the neural network is run**

The plot in figure 17 shows the convergence of the mean-square error versus the number of iterations needed to train neural network. This network has 121 inputs with a network structure of 15, 21, 11. The bias input of each neuron is one and alpha of the activation function is 0.8:Phi$(v) = \frac{1}{1+e^{(-\alpha v)}}$. Also, the momentum value is set to 0.3. This momentum value was held constant throughout the training of the neural network, but could be increased to increase the rate of convergence. When training the network I increased the momentum value from 0.3 to 0.9 after the network was able to correctly classify all input training images with a mean-square error less than 0.1 [2]. This change in momentum value causes a jump in the mean-square error convergence plot which can be examined in figure 18. Momentum is increased to 0.9 to speed up the convergence of the mean-square error (mse) where mse $= \frac{1}{3}\sum_{i=1}^{3}(d_i - y_i)^2$ and d is an output value from the neural network and y is the desired output. It is divided by 3 since there are 3 output neurons to the network. Both the momentum value and alpha where found by experimentation.

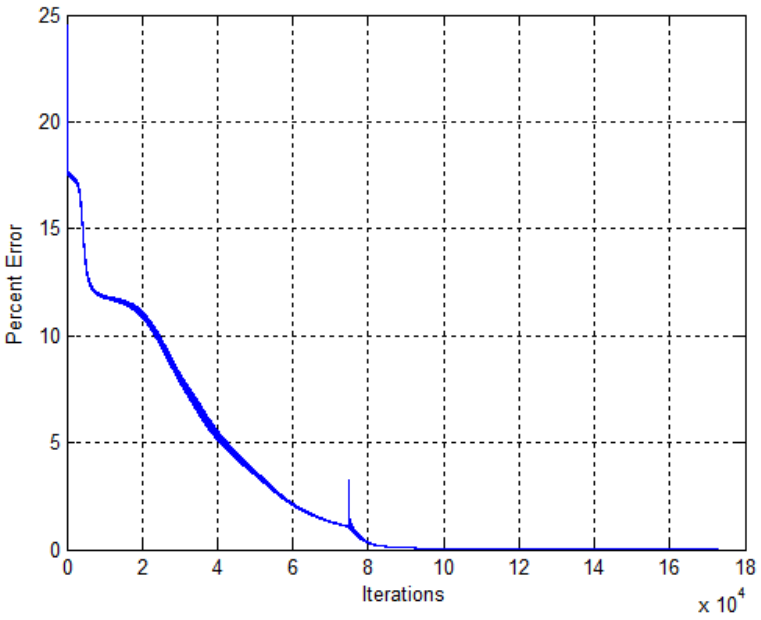



**Figure 18: Mean squared error convergence as the neural network is run**

By altering the momentum while the network is being trained it was able to train to a mean square error of 0.01 faster. While maintaining a momentum coefficient of 0.3, it took the network 690,000 iterations versus 160,000 iterations when changing the momentum coefficient from 0.3 to 0.9. The only drawback is the blip in the mean squared error plot caused by the change of momentum, which can be seen around iteration 75,000 in figure 18. For further optimization, the momentum coefficient can be adjusted multiple times throughout the training algorithm, for example the momentum could be increased at different thresholds of the mean squared error [16].

As it was mentioned previously, the activation function is logsig, which is shown in figure 19. This logsig activation function has an output between 0 and 1 depending on the input. The reason this activation function is used is because the output of the neural network is in binary form, so either a zero or a one. The slope of the function in the range between zero and one is varied based on the alpha value, shown in figure 19. The alpha value of the neural network used in this paper is 0.8. This value was found based

on trial and error on what causes the neural network to converge to a mean square error of 0.01 faster, figure 20.

$$\mathrm{Phi}(v) = \frac{1}{1+e^{(-\alpha v)}} \quad \textbf{(3)}$$

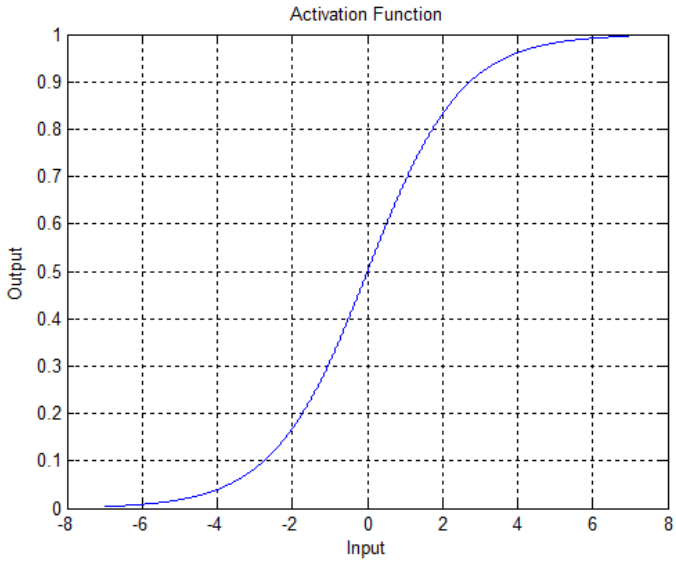



**Figure 19: Activation Function used in the neural network**

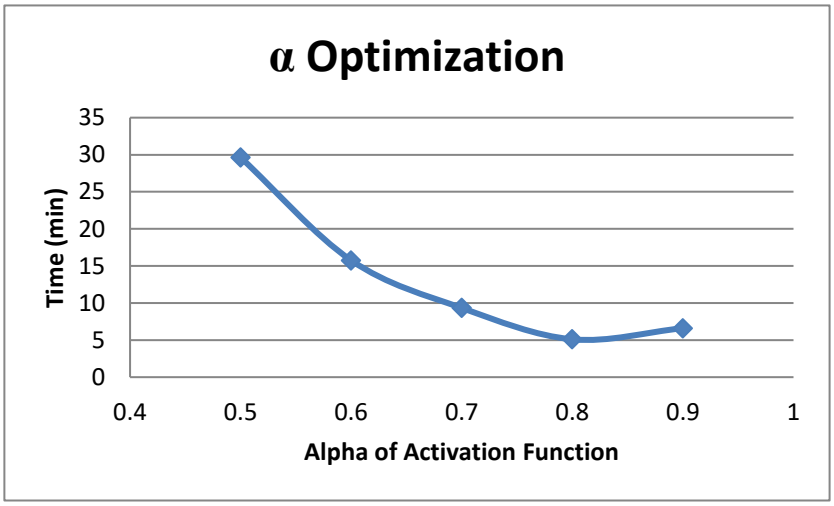



**Figure 20: Time of Neural Network convergence for differing alpha values**

The final constant that needs to be optimized is eta (η). This constant comes into play when updating the weights of the neural network. The equation is as follows (equation 4).

$$\Delta w_i = \eta(\text{error}) + \gamma w_{i-1} \quad \textbf{(4)}$$

It can be seen that as eta increases the more the weights of the neural network are changed from iteration to iteration.

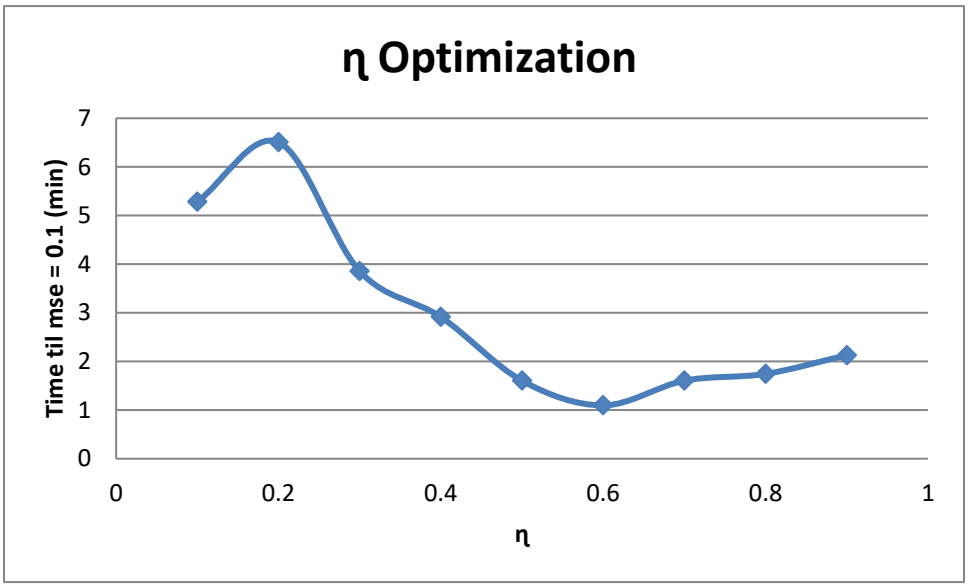



**Figure 21: Time it takes for convergence with differing η (eta) values**

Based on all the trial and error iterations from momentum, alpha, size of neural network and eta the final neural network is shown in figure 22.

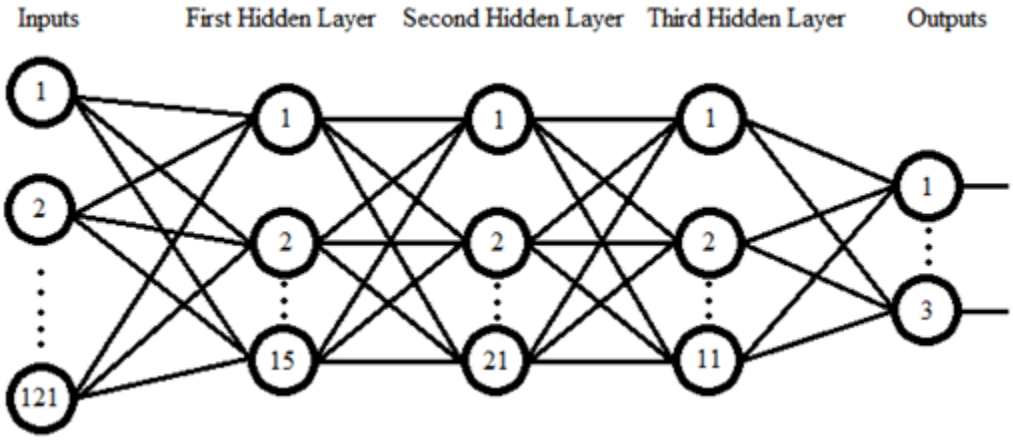



**Figure 22: The optimum Neural Network for speed of convergence based on trial and error**

The alpha, $\alpha = 0.8$, momentum, $\gamma = 0.3$ and increases to $\gamma = 0.9$ after the mean squared error gets below mse = 0.1, and the eta, $\eta = 0.6$. These optimizations were performed with a network setup of 121 inputs and 3 outputs, and will slightly change if the number of inputs or outputs increase or decrease, or the training inputs change. The formula to update the weights is shown above in equation 4. As seen in figure 22, there are three outputs to the tested neural network. Binary encoding was chosen as the output to the neural network because it increases the total possible number of outputs while decreasing the number of connections needed by the network.

DISPLAYING RESULTS:

**TABLE 1: Confusion matrix for Fourier Transform with rotational distortion**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 0 | 18 | 0 | 15 | 0 | 1 |
| **2** | 7 | 20 | 0 | 3 | 1 | 3 |
| **3** | 0 | 1 | 17 | 0 | 16 | 0 |
| **4** | 0 | 12 | 0 | 18 | 0 | 4 |
| **5** | 0 | 3 | 5 | 9 | 16 | 1 |

A detailed way to display the performance of the neural network is with a confusion matrix [2] shown in Table 1. Each column represents the output classified by the neural network while the rows represent the test case input into the neural network. A perfect confusion matrix will have zeroes everywhere but along the diagonal. The numbers along the diagonal, in Table 1, mean the network correctly classified the input image. The confusion matrix of Table 1 shows that the overall correct classification is 42%. In the confusion matrix the other column represents values form the network that are not in the range of the number of individual eyes. Since we are using the binary encoding with three output neurons there are a possible of 8 output combinations ($2^3$), but are only using 5 eyes so there are 3 output values that do not correspond to the expected output. These are classified as “other”. Each row adds up to 34 because we are testing with 34*5=170 rotated test images so ideally (100% correct classification) there would be

a 34 in each square along the diagonal. The 34 images are made up of 2 images from each of the 5 people with a different rotation applied from the range of -16 to +16 degrees. For example, 1 person would contribute 2 images and each image would be rotated 17 times ($-4^o$, $-2^o$, $0^o$, $2^o$, $4^o$, etc.).

TESTING NEURAL NETWORK ROBUSTNESS

To examine the robustness of the coefficients as representations of the irises, noise is applied to the test images. Noise and distortion will be expected with any camera or image of an eye. The amount of noise in an image can be measured with a signal to noise ratio (SNR). The SNR of the noisy image is the ratio of the signal power to the added noise power [8], and is calculated by

$$\text{SNR} = \frac{\text{Image Power}}{\text{Noise Power}} = 10 * \log\left(\frac{\sum_{x=1}^{N*M} I(x)^2}{\sum_{x=1}^{N*M} [I(x) - \text{Noise}(x)]^2}\right) \quad \textbf{(5)}$$

In equation 5, *I*, represents the original eye image intensity as a column vector and *Noise* represents the same eye image but with noise applied to it. Also, N is the number of columns in the image and M is the number of rows.

Rotational distortion occurs when multiple images of the same eye are captured, not all have the camera and the eye perfectly aligned. In this thesis, I assumed that the maximum amount of rotation would be plus and minus 16 degrees. An example of an input image to the neural network is shown in figure 23.

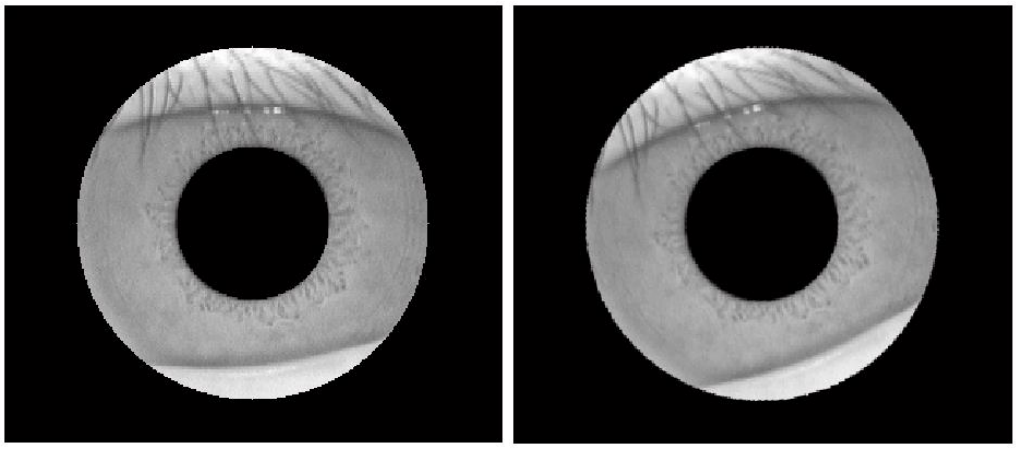


**Figure 23: (a) Eye image after iris is detected (b) eye image after iris detection with a $15^0$ rotation**

Each test eye is rotated from negative sixteen degrees to sixteen degrees and then input into the neural network. The output of the network is examined to see if the network (trained with clean images) was able to correctly classify the test eye image with rotation.

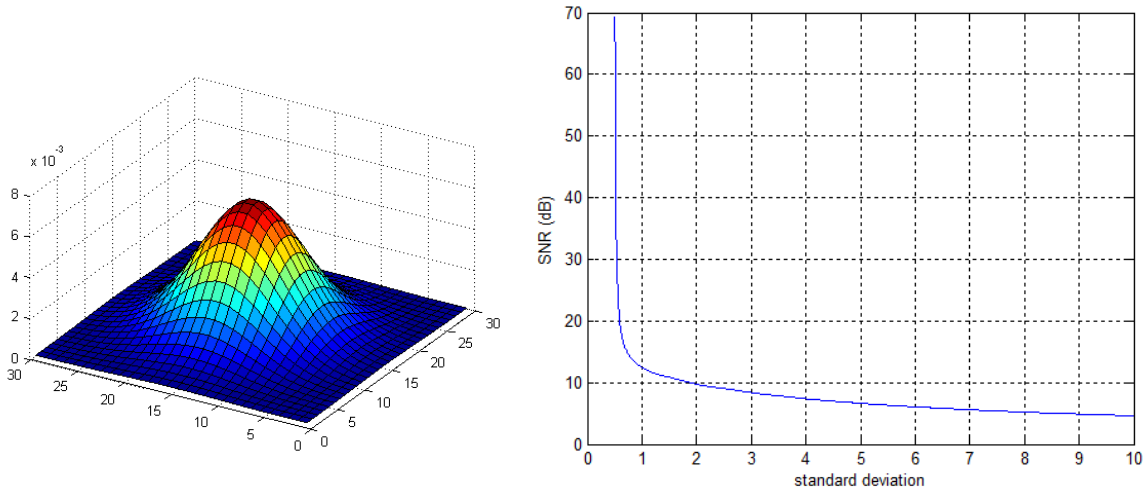



**Figure 24: (a) 3D representation of a Gaussian kernel (b) plot representing its effect on the SNR on the image in Figure 23a**

The second condition applied to the test images is Gaussian blur. After the neural network was trained with 5 images per eye, each test image has Gaussian blur applied to it. These two images are convolved with a Gaussian kernel of various standard deviations and an example is shown in figure 24. The Guassian kernel example in figure 24 is created in Matlab with the function call "fspecial('gaussian', 30, 5)." The mathematical equation for a Gaussian kernel is depicted in equation 6, where $\sigma^2$ is the desired standard deviation.

$$G_{\text{kernel}}(x, y) = \left(\frac{1}{2\sigma^2}\right) e^{\frac{-(x^2+y^2)}{2\sigma^2}} \qquad \textbf{(6)}$$

It is important that the sum of all values (x, y), representing all pixels, in the Gaussian kernel sum to 1. This assures that the original image does not become brighter or darker, it only becomes blurred. The image in figure 25 shows the effect of Gaussian

blur on an iris image. As the kernel size increases the signal to noise ratio decreases. The images in figure 25 show the effect of having a Gaussian blur applied to an image.

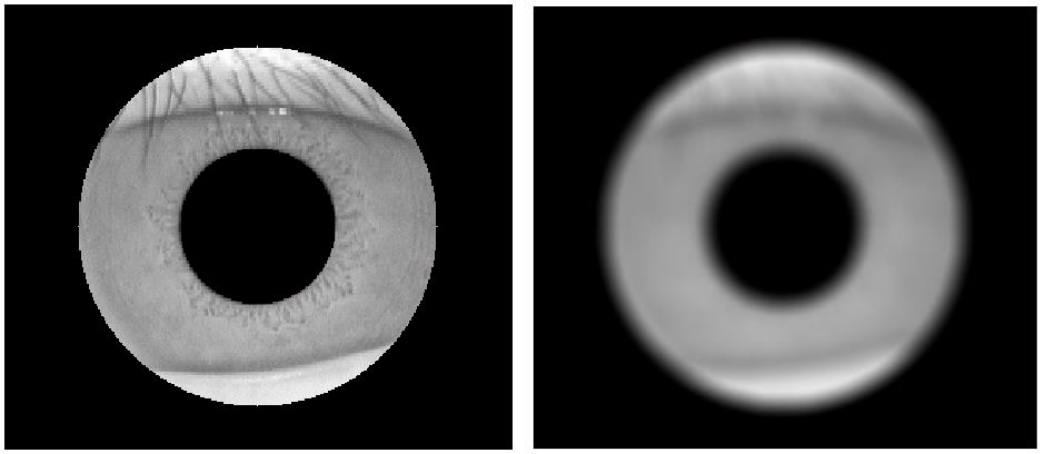
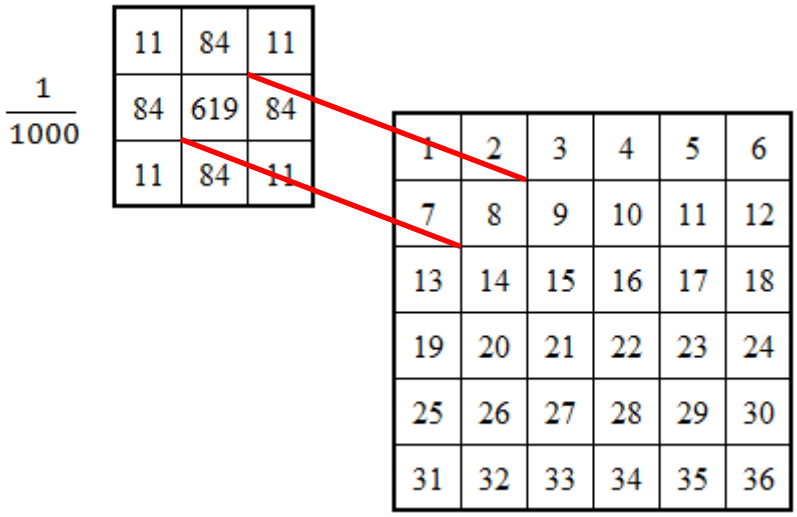


**Figure 25: Example of an eye image with Gaussian blur applied (SNR = 5)**

The process of 2 dimensional convolutions is done by multiplying the Gaussian kernel with every pixel of the original image. The new pixel value is then the weighted sum of all the surrounding pixels. The number of surrounding weights used in the weighted sum depends on the size of the Gaussian kernel. Figure 26 is an example of a 3 by 3 Gaussian kernel being convolved with an image and shows a single calculation of the convolution and the convolution is done to every pixel in the image to create a blurry image. A more blurry image can be created in two ways; first the size of the Gaussian kernel can be increased or second, the standard deviation of the kernel can be increased. For this project, I chose to change the size, from 1 pixel up to 10 pixels, of the kernel to increase the blurriness (decrease the signal to noise ratio). The standard deviation ($\sigma^2$) stayed at 0.5.




$$I(2,2) = \frac{1}{1000}[(1*11)+(2*84)+(3*11)+(7*84)+(8*619)+(9*84)+(13*11)+(14*84)+(15*11)]$$

$$= 7.99$$

**Figure 26: Convolution Example with a 3 by 3 Gaussian kernel**

Different sizes of Gaussian kernels are applied to the eye images resulting in images with different signal to noise ratios. The noisy, blurry, images had their features extracted and the coefficients were input into the pre-trained neural network and tested for robustness. The edges were dealt with by mirroring them to adjacent pixels. This helps by not assuming black pixels on the border which would decrease the intensity of blurred pixels along the edges

The third case is to apply salt and pepper noise to the test images. Salt and pepper noise are randomly distributed white and black pixels throughout the image, as seen in figure 27. This noise would occur if the camera has a mechanical problem or a digital conversion problem [19]. Either individual pixels in the camera are dead or the sensing of the amount of light received could be malfunctioning, and with so many pixels in a camera, this error is bound to occur. The SNR can be measured (Equation 5) by

calculating the signal to noise power ratio between the original image (assumed to have little to no noise) and the new image that has had noise applied to it, figure 27. After the neural network is trained with 5 images per eye, various degrees of salt and pepper noise (SNR ranging from 5dB to 30dB) is applied to the two test images and then the output of the network is examined to see what percentage of the noisy images are correctly classified.

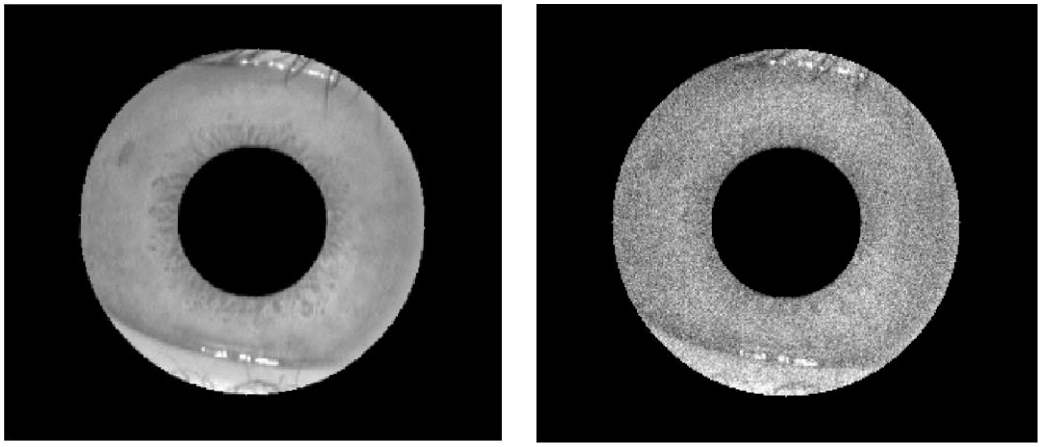


**Figure 27: (a) original image with no noise (b) same image with salt and pepper noise with SNR =7dB**

FOURIER TRANSFORM

The Fourier transform of an image contains frequency information in its coefficients. Each coefficient represents the amplitude and phase of a frequency component (only the magnitude is used as an input to the neural network).

$$I(u,v) = \sum_{x=1}^{M} e^{-j2\pi ux/M} \sum_{y=1}^{N} f(x,y)\, e^{-j2\pi vy/N} \quad (7)$$

The Fourier transform is a summation of real cosine, and imaginary sine functions with different frequencies, with each frequency assigned a weight [8]. These assigned weights are used as the inputs to the artificial neural network. In equation 7, M is the number of rows in the image, N is the number of columns in the image, I(x, y) is the intensity at pixel location (x, y), (i.e. x rows down and y columns to the right, with pixel (1,1) in the upper left of the image). Also in equation 7, u and v are the pixel locations in the frequency domain image, which is the same size as the image having the Fourier transform applied. An example of an image and its Fourier transformed image are shown in figure 28.

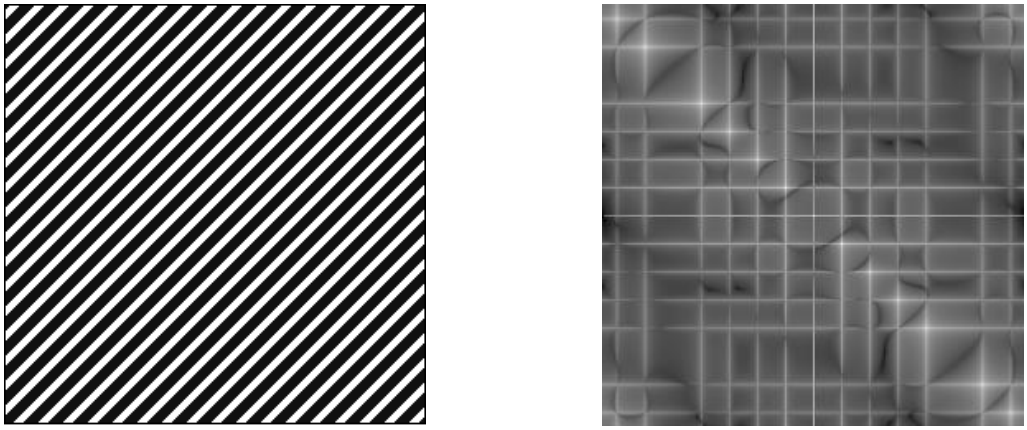


**Figure 28: (a) original image (b) 20log*fft(original image)**

As an example, the original image in figure 29a has its Fourier transform taken and then has a $\log_{10}$ function applied to it, to decrease the dynamic rage (difference between largest and smallest intensity value of the image). The original image has its biggest frequency component along the diagonal and as can be seen in the Fourier transformed image, the largest (brightest points) come along the same diagonal as the stripes of figure 28a. In a more complex image, the Fourier transform has more information concentrated in the lower frequency (center of Fourier image) and is shown in figure 29b.

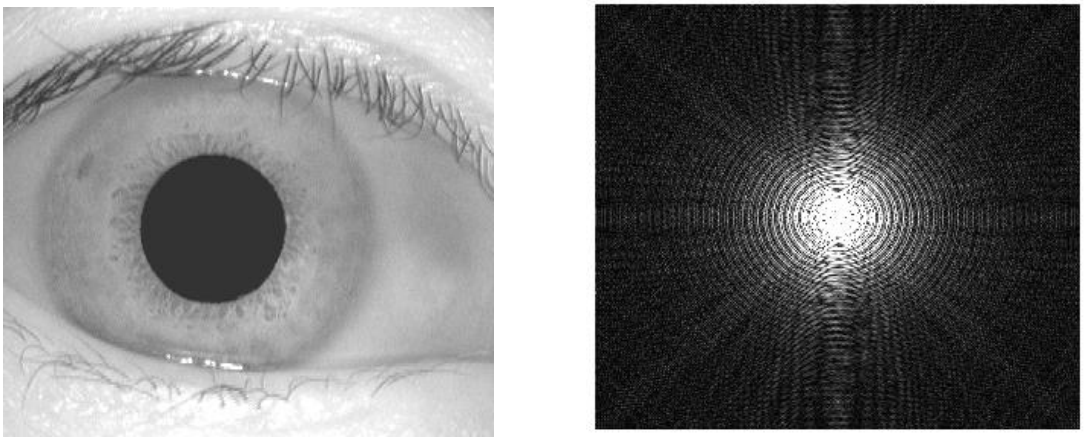


**Figure 29: (a) Eye image and (b) its Fourier transform**

The image of figure 29b represents the Fourier transform of the eye image of figure 29a. Most of the frequency components are located in the center of the Fourier transform image, meaning that the majority of the frequency components are low frequency. The center pixel in the Fourier transform image is the DC value and the greater the distance from a pixel to the center of the Fourier image, the higher the frequency. The fact that the majority of frequency components are small can be used to decrease the number of values needed to represent an image. Instead of inputting every pixel value of the original image into the neural network, the center pixel values of the

Fourier transform image can be used as a representation of the original eye image. This is depicted in figure 30.

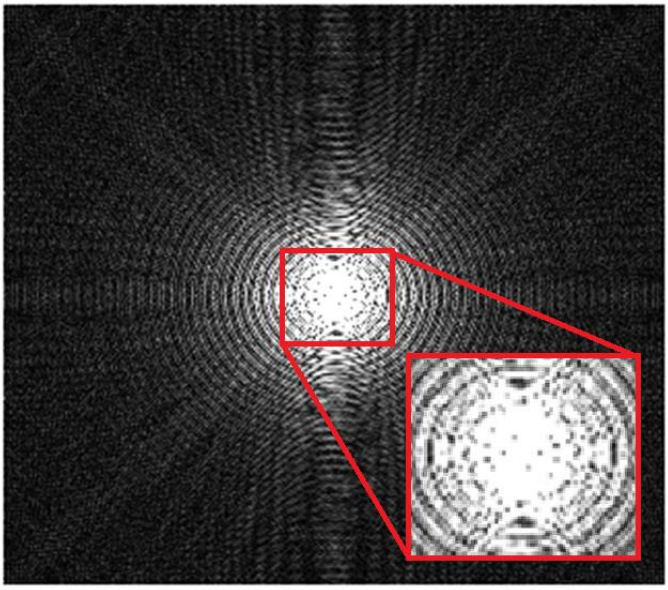


**Figure 30: Depiction of highest weighted frequencies**

The Fourier (frequency domain) image in figure 30 shows that the largest (brightest) weights are in the lower frequencies. These coefficients will eventually be the inputs to train the neural network; but first, normalization must be performed [10]. Normalization is required because the dynamic range on the raw Fourier transform image is huge, the maximum pixel value is 10,267 and the minimum is -8,674. The large range in values makes it almost impossible for the neural network to converge. Also, the negative value come into play here because the Fourier transform coefficients of the masked image are subtracted from the iris image. To improve the neural network convergence, a normalization of the images' Fourier coefficients is applied.

$$\tilde{\imath}u = \frac{I_u - I_{min}}{I_{max} - I_{min}} \quad \textbf{(8)}$$

In normalization equation 8, u is the number of coefficient, ranging from 1 to MN where MN is the total number of pixels (M rows * N columns), $\tilde{\imath}u$ is the updated

coefficient, $I_u$ is the original coefficient, $I_{min}$ and $I_{max}$ are the smallest and largest coefficients based on the Fourier transform of 700 different eye images. The normalization leaves the range of coefficients between 0 and 1. This is shown in figure 31b where in the first image (original coefficients) the largest appears around 6000 and the smallest around -6500, but after normalization all the coefficients are between 0 and 1. The reason the highest coefficient is $\neq 1$ is that $I_{min}$ and $I_{max}$ are calculated from all inputs to the neural network and applied to all individual inputs during normalization procedure, and not each individual image. It is important to have all eyes, individually normalized according to the groups minimum and maximum to insure consistent normalization. This can become an issue if a new, known, image is added to the set and has a larger or smaller value than the previous minimum or maximum. If this occurs, new coefficients must be obtained for all images.

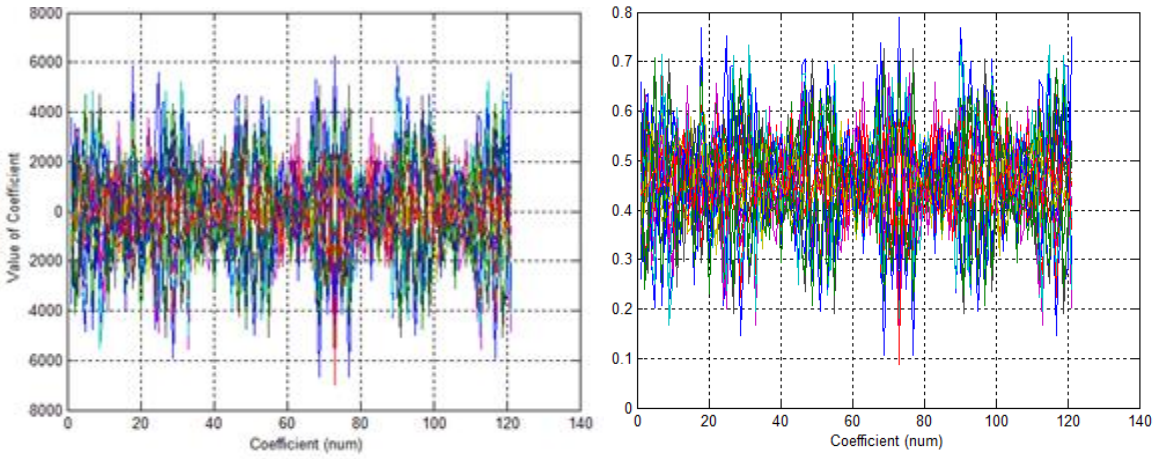



**Figure 31: (a) Original Fourier coefficients and (b) coefficients, after normalization**

FOURIER TRANSFORM RESULTS

After training with clean images, Fourier Transform coefficients from clean test images, classify images with 50% accuracy. Results, with noise and distortion are applied to inputs are summarized below.

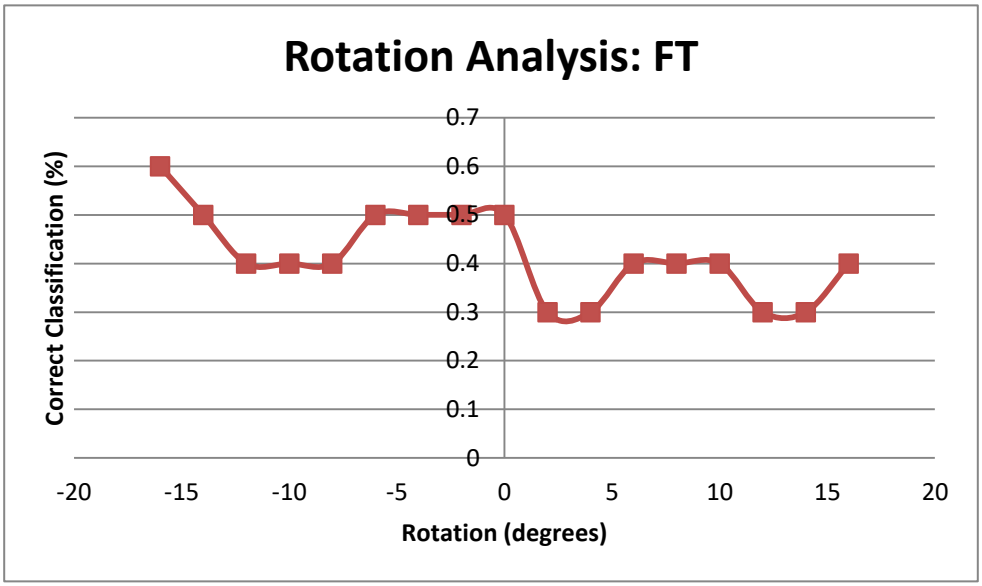



**Figure 32: Plot of the robustness of Fourier representation with added rotation distortion**

**TABLE 2: Confusion matrix for Fourier Transform with rotational distortion**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 0 | 18 | 0 | 15 | 0 | 1 |
| **2** | 7 | 20 | 0 | 3 | 1 | 3 |
| **3** | 0 | 1 | 17 | 0 | 16 | 0 |
| **4** | 0 | 12 | 0 | 18 | 0 | 4 |
| **5** | 0 | 3 | 5 | 9 | 16 | 1 |

The test concludes, with results shown in figure 32 and Table 2 that with Fourier Transform inputs to the neural network and up to a plus and minus sixteen degrees rotation, it has an overall correct classification percentage of 42%, calculated below.

$$\frac{(0 + 20 + 17 + 18 + 16)}{170} * 100 \quad = \ 41.76$$

Different signal to noise ratios ranging from 5dB to 30dB, due to salt and pepper noise, are applied to the test images and the results of correct classification are summarized in figure 33.

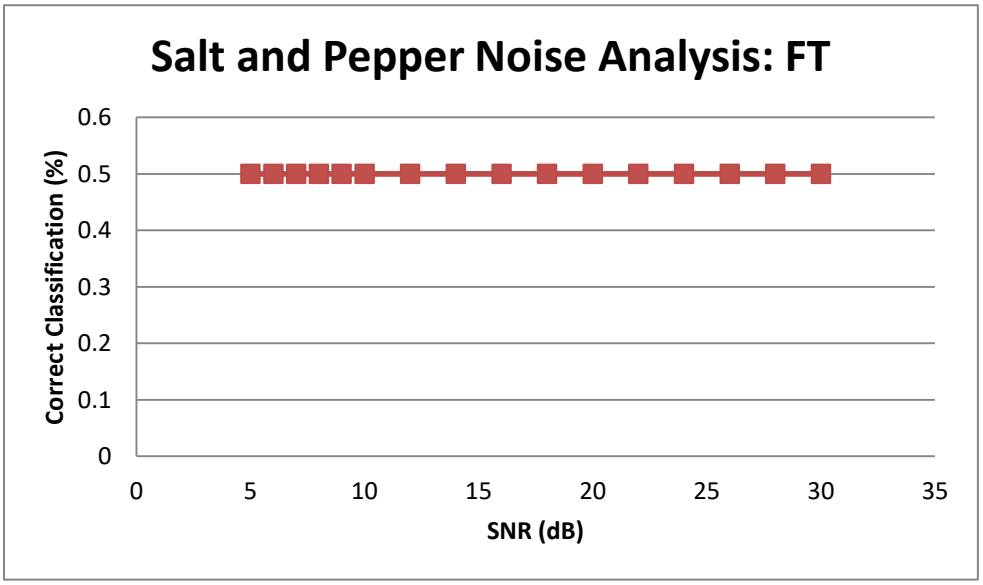



**Figure 33: Summary of Salt and Pepper noise analysis with FT coefficients**

The same test examined in figure 33 is done but more details are saved with a confusion matrix, Table 3. Both the confusion matrix and plot of figure 33 show that no matter what the amount of salt and pepper noise, the neural network classifies correctly at a 50 percent rate.

**TABLE 3: Confusion Matrix for salt and pepper test cases with SNR from 3 to 30 dB**

| | 1 | 2 | 3 | 4 | 5 | other |
|---|---|---|---|---|---|---|
| **1** | 0 | 17 | 0 | 17 | 0 | 0 |
| **2** | 0 | 34 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 17 | 0 | 17 | 0 |
| **4** | 0 | 0 | 0 | 17 | 0 | 17 |
| **5** | 0 | 0 | 17 | 0 | 17 | 0 |

Different sizes of Gaussian kernels are applied to the test images resulting in images with different "signal to noise ratios" (SNR). The blurry images had their Fourier transforms taken and the coefficients were input into the pre-trained neural network and tested for robustness. Results of the Gaussian blur analysis are shown in figure 34.

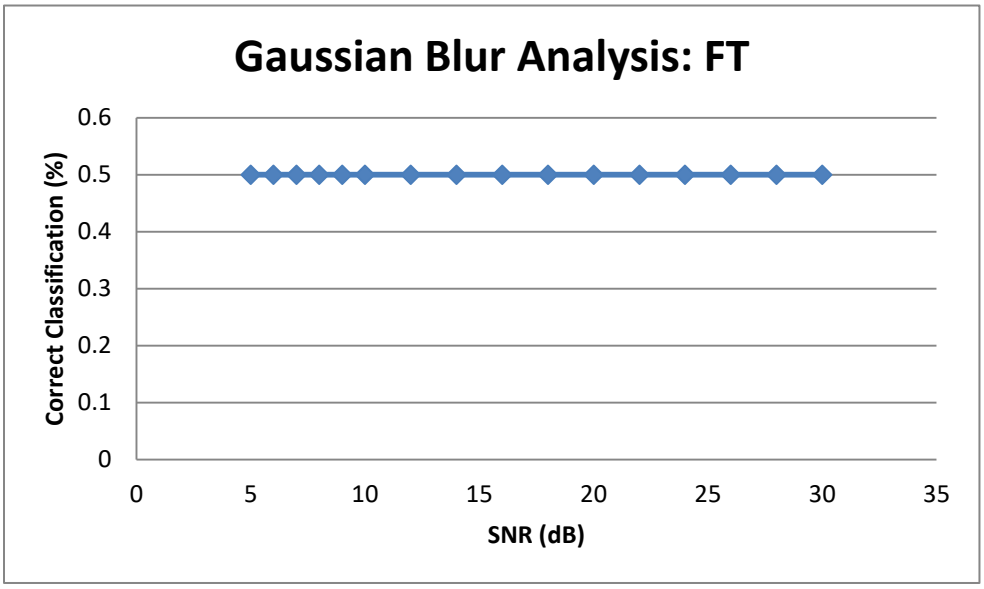



**Figure 34: Summary of Gaussian blur analysis with FT coefficients**

The same test examined in figure 34, is done but more details are saved with a confusion matrix, Table 4. Both the confusion matrix of table 4 and plot of figure 34 show that no matter what the amount of Gaussian blur, the neural network classifies correctly at a 50 percent rate.

**TABLE 4: Summary of Gaussian blur analysis with FT coefficients**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 0 | 17 | 0 | 17 | 0 | 0 |
| **2** | 0 | 34 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 17 | 0 | 17 | 0 |
| **4** | 0 | 0 | 0 | 17 | 0 | 17 |
| **5** | 0 | 0 | 17 | 0 | 17 | 0 |

EIGEN EYES

A method for iris identification is using Eigen eyes (principal component analysis, PCA). Finding the Eigen coefficients is a type of PCA process and is typically used in facial recognition [8].

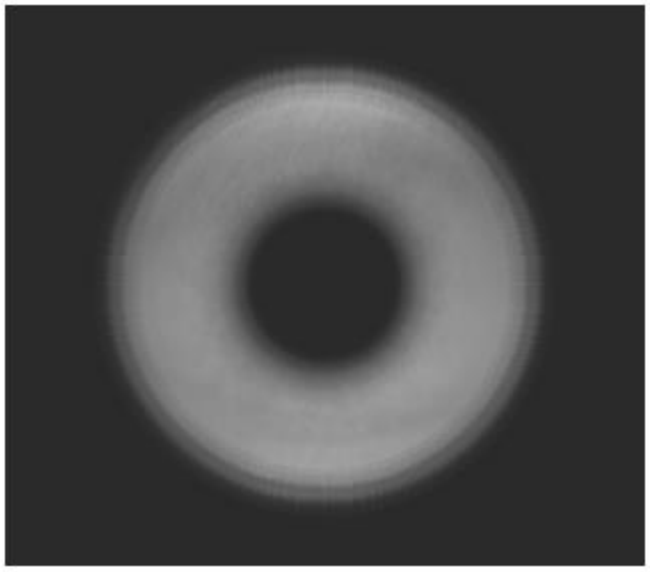


**Figure 35: Mean Eye used to calculate Eigen features**

The concept of the Eigen space is to find the average eye, as shown in figure 35, by using a linear combination of a set of sample eyes. Each component of the average eye will be a weighted sum of the eyes in the sample space. Once the components of the average eye are found, any new eye can be represented as a weighted sum of those standard eye components. Thus, an eye can be represented as the weighted values of each component, creating a simple feature space to use. This feature space is used to classify eyes and reduce the memory requirement needed to store individual eye information. An oversimplified example of this is shown in figure 36. In this research, we will use a combination of 50 eyes to create the average eye.

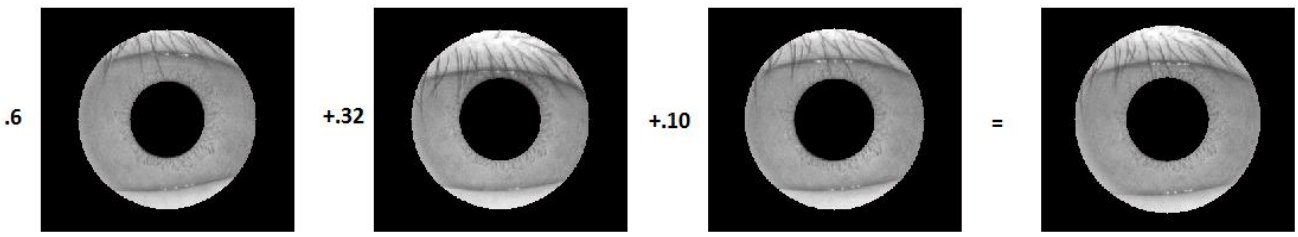



**Figure 36: Shows how Eigen features are similar to a linear combination of different eyes**

The mathematical calculations needed in order to calculate the average eye and any new eye's weights are simple and straightforward [8]. First, all 50 sample eyes, each represented as an M x N matrix (M is the number of rows and N is the number of columns in the image), are transformed into (M*N) x1 column vectors called gamma (Γ). With these 50 column vector representations of the iris images, the mean face, Ψ, is found by finding the average Γ vector. Each Γ vector is then normalized by having the mean subtracted from it; resulting in a (M*N) x1 vector, φ. We need to calculate the eigenvectors *u* of the covariance matrix, C, which is defined in equation:

$$C = \frac{1}{P}\sum_{n=1}^{P} \varphi_n \varphi_n^T = AA^T, \qquad where\ A = [\varphi_1 \varphi_2 \ldots \varphi_P] \qquad \textbf{(9)}$$

In equation 9, C is the covariance matrix of φ, and is of size (M*N) x (M*N) and A is a concatenation of 50 normalized vectors giving it a size of (M*N) x 50. Also, P is the number of eyes being used to create the linear combination, in this case, 50.

However, the result of this calculation is too large (M*N) x (M*N), which with 280x320 images has over 7 trillion pixels. Instead, $A^TA$ is used to results in a matrix of size 50x50. It is important to note that, the Eigen vector of $A^TA$, *v* is not the same as *u*, but is linearly related. The eigenvectors *u* can be found from *v* by :

$$u_i = Av_i \qquad \textbf{(10)}$$

The significance of each eigenvector is lower as the eigenvalue decreases. Only the K best eigenvectors (i.e. the ones with the largest Eigen values), should be kept.

In order to find the weights, *w*, of a new eye ($\varphi_{new}$ is the new eye minus the mean eye of the 50 eyes, ψ) using the K eigenvectors from above, the following, equation 11, is used:

**(11)**

$$\varphi_{new} - \psi = \sum_{i=1}^{K} w_i u_i, \qquad (w_i = u_i^T \varphi_{new})$$

The weights, $w_i$, can then be used for identifying and classifying an eye. In this thesis, 50 eyes from the CASIA database were used to find the mean eye and the largest K=50 weights were saved for classification. None of the eyes used create the mean eye are used to train or test the neural network.

Although the Eigen space method is typically used for facial recognition, it can be easily adapted to work with images of any object that can be used for identification. After calculating the average eye from the set of sample eyes, a data set was compiled with multiple samples of several peoples' eyes. The eye images are projected onto the Eigen space and the 50 most significant weights were calculated. These weights were fed into a neural network as input for training. The neural network selected for this thesis is a feed forward network with the Levenberg-Marquardt back propagation learning algorithm. The size of the network is determined empirically through trial and error.

EIGEN TEST RESULTS

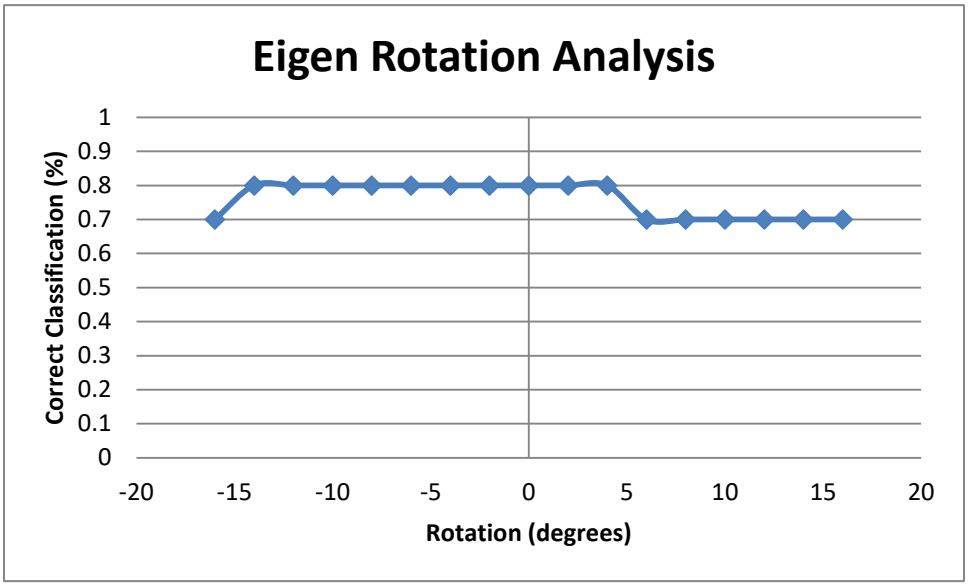



**Figure 37: Analysis of rotation with Eigen inputs**

The graph in figure 37 shows that the maximum degradation of the neural network's correct classification is ten percent. That is, normal classification is 80% and rotation may decrease the network classification accuracy to 70%. To get this data 10 test images were rotated from -16 to 16 degrees (every 2 degrees).

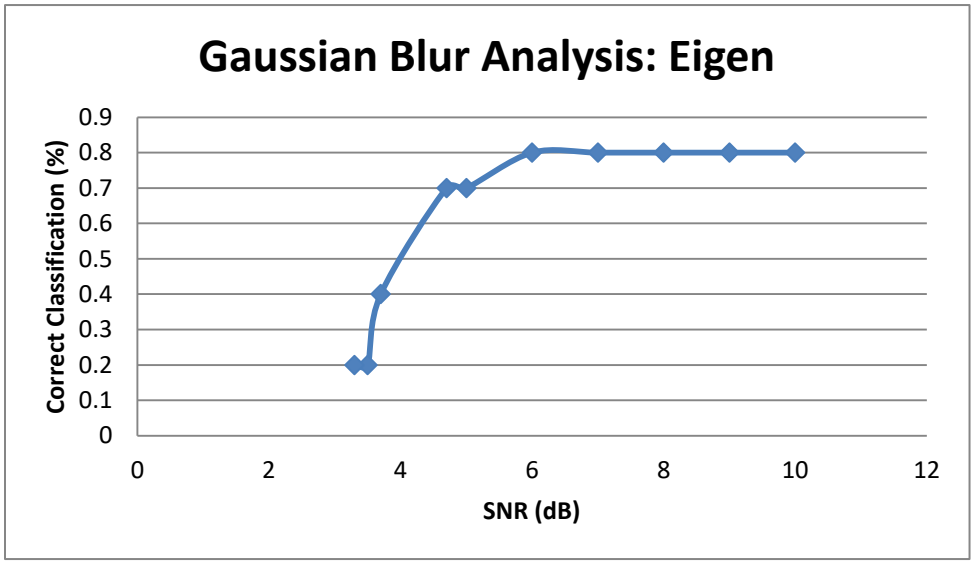



**Figure 38: Gaussian Blur analysis for Eigen coefficients**

Each test image had different amounts of noise applied to them and the results are summarized in figure 38 and 39.

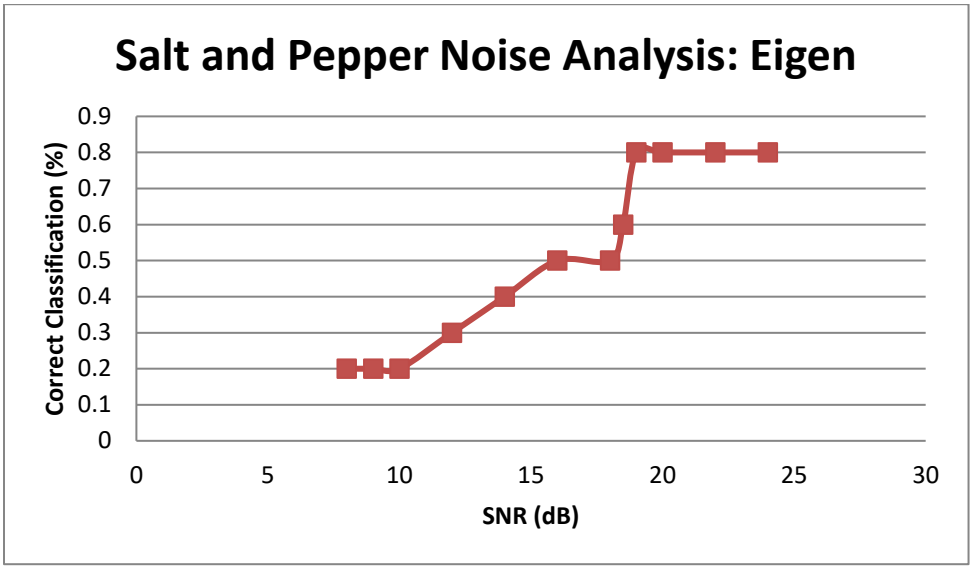



**Figure 39: Salt & Pepper Noise analysis for Eigen coefficients**

Based on the results of figure 38 and 39, the neural network is more susceptible to salt and pepper noise than Gaussian blur. This is shown in the plot because the images with salt and pepper noise start to decrease in classification percentage at a SNR of 19dB while images with Gaussian blur start to degrade at and SNR of 6dB. That is, more Gaussian blur needs to be added to an image than salt and pepper to disrupt the classification of the neural network.

There are advantages and disadvantages to the Eigen feature space versus the Fourier transform feature space. Most importantly, an iris can be represented with a significantly smaller number of coefficients with Eigen weights for similar classification rate. The Fourier transform feature space requires thousands of coefficients, whereas, the Eigen feature space requires only 50 (for this test), to represent a complete iris. Fewer coefficients to completely represent an iris is an advantage because it means fewer inputs

to the neural network, which will lead to shorter training times and fewer neurons. Another advantage of the Eigen feature space is that it can better deal with changes in lighting conditions, because in the creation of the average eye, the average lighting condition is used. A disadvantage of the Eigen feature space is that it has a more difficult time dealing with rotations. With preprocessing of the iris, it is centered in the image, meaning that with some rotation the coefficients that make up the Eigen space can dramatically change. Whereas, since the iris is centered in the image, the Fourier transform coefficients of the original and rotated image will be fairly similar.

DISCRETE COSINE TRANSFORM

Another way to reduce the number of values needed to represent an image is to use the discrete cosine transform (DCT). The DCT transforms an image from the spatial domain to the frequency domain. Most of the image's information will be represented by DCT coefficients of lower frequency components (located in the upper left of the DCT coefficient matrix). In order to reduce the number of coefficients to represent an image the coefficients representing the higher frequency components are ignored [8].

Similar to the Fourier transform, the discrete cosine transform is a sum of cosine waveforms that oscillate at different frequencies. The largest difference between the Fourier transform and the DCT is that the DCT coefficients are all real numbers, and it has been found that using strictly cosines is more efficient than using sine waveforms as in the Fourier transform. The formula used to transform the original eye image, $f(x,y)$, where $f(x,y)$ is the intensity of the original eye image, into its frequency representation matrix, $I(u,v)$:

$$I(u,v) = \alpha_u \alpha_v \sum_{x=1}^{M} \sum_{y=1}^{N} f(x,y) \cos\left(\frac{\pi(2x+1)u}{2M}\right) \cos\left(\frac{\pi(2y+1)v}{2N}\right) \quad \textbf{(12)}$$

Where M is the number of rows and N is the number of columns in the original image. Also

$$\alpha_u = \begin{cases} \frac{1}{\sqrt{M}}, & u = 1 \\ \sqrt{\frac{2}{M}}, & 2 \le u \le M \end{cases} \qquad \alpha_v = \begin{cases} \frac{1}{\sqrt{N}}, & v = 1 \\ \sqrt{\frac{2}{N}}, & 2 \le v \le N \end{cases} \quad \textbf{(13)}$$

The DCT results in a matrix the same size as the original eye image (280 x 320), and an example is shown in figure 40. As can be seen in figure 40, the lower frequency coefficients (upper left) have larger, more powerful, coefficients (brighter pixels).

**Figure 40: Matrix of DCT coefficients**

Since the DCT coefficients represent the spatial frequencies in the eye images and the iris is centered in the image, a rotation to the iris should not drastically affect the coefficients. The coefficients that are used to train the neural network come from a square in the upper left corner of the DCT coefficient matrix. The 121 DCT coefficients are taken from an eleven by eleven pixel box in the upper left corner of the DCT coefficient matrix, figure 41. The robustness of the DCT as a classifier is examined by applying noise to the test images and seeing how well the trained neural network can perform.

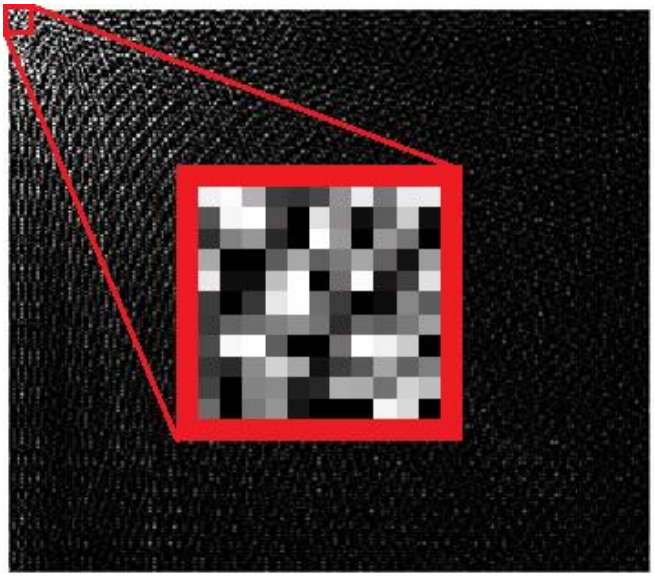


**Figure 41: DCT coefficients chosen to represent an iris**

DISCRETE COSINE TRANSFORM RESULTS

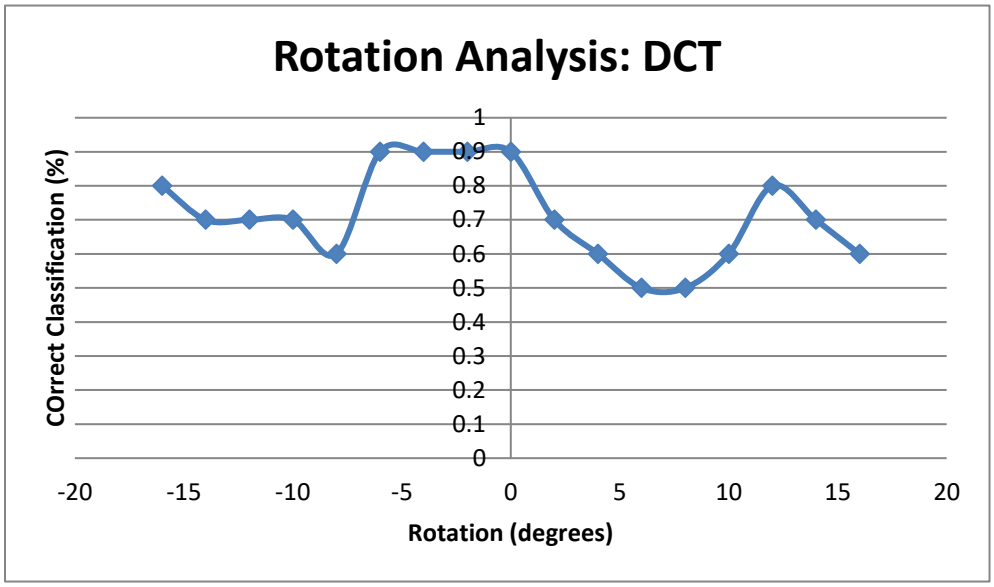



**Figure 42: Rotation Analysis performed on neural network with DCT inputs**

Figure 42 shows how the neural network reacts to rotation up to plus and minus 16 degrees. It also shows that without any rotation or noise the classification rate of the neural network is 90%.

**TABLE 5: Confusion Matrix for test cases rotated from -16 to 16 degrees**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 15 | 0 | 0 | 11 | 0 | 16 |
| **2** | 0 | 26 | 0 | 15 | 0 | 1 |
| **3** | 4 | 0 | 38 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 42 | 0 | 0 |
| **5** | 0 | 0 | 3 | 11 | 28 | 0 |

Different signal to noise ratios ranging from 3dB to 30dB are applied and the results of correct classification are summarized in figure 43.

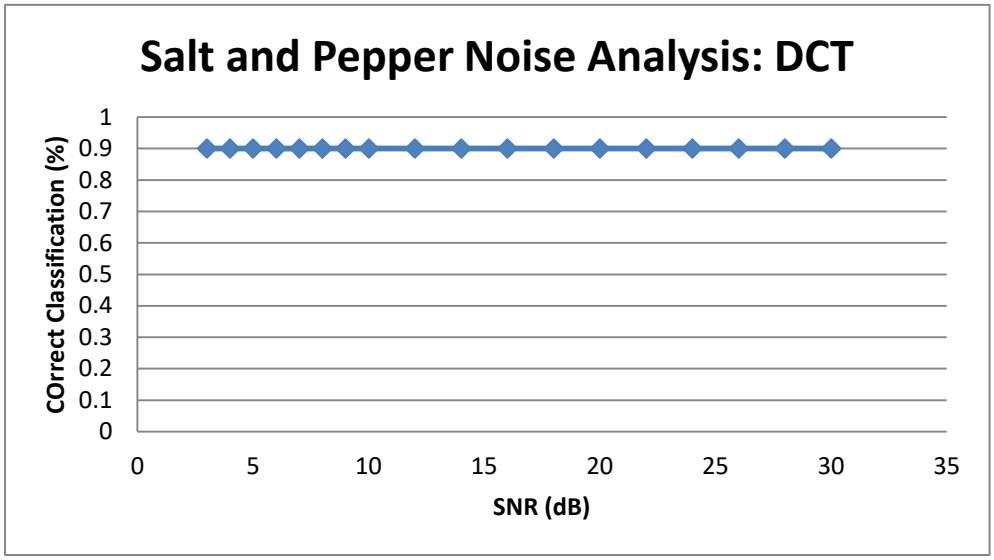



**Figure 43: Summary of Salt and Pepper noise analysis with DCT coefficients**

The only error that is shown in the confusion matrix is that the test eye 1 has a false positive classification, which leads the network to have a false positive classification rate of 10 percent $\frac{18}{180} * 100 = 10\%$.

**TABLE 6: Confusion Matrix for salt and pepper test cases with SNR from 3 to 30 dB**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 18 | 0 | 0 | 18 | 0 | 0 |
| **2** | 0 | 36 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 36 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 36 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 36 | 0 |

Different sized Gaussian kernels, are applied to the test images before they are put through the trained neural network. Results of this noise analysis are summarized in figure 44.

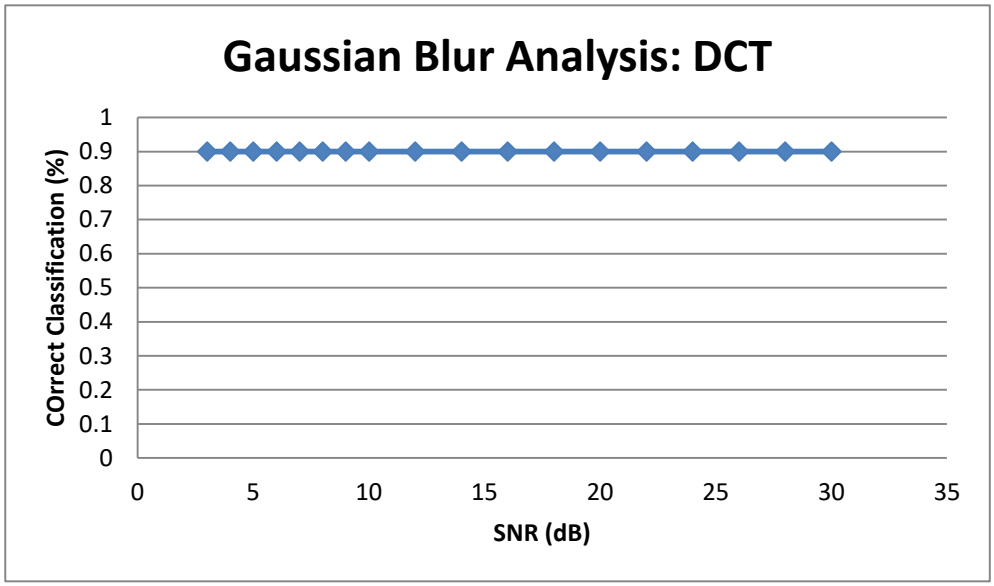



**Figure 44: Gaussian Blur analysis plot with DCT inputs**

Just like the test images with salt and pepper noise applied, the only error is the first image is incorrectly classified as the fourth image. This leads the network to have a false positive classification, and error, rate of 10 percent $\frac{18}{180} * 100 = 10\%$..

**TABLE 7: Confusion matrix with DCT inputs and added Gaussian blur**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 18 | 0 | 0 | 18 | 0 | 0 |
| **2** | 0 | 36 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 36 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 36 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 36 | 0 |

Based on the noise simulations performed, the discrete cosine transform coefficients perform better with salt and pepper or Gaussian blur applied to the test images than when the image is rotated. This is due to the fact that salt and pepper noise and Gaussian blur both effect high frequency components in the image and since only the upper left corner of the DCT coefficient matrix (low frequency components) the noise does not affect the coefficients used to represent the test images.

WAVELET TRANSFORM

Another way to decrease the number of coefficients needed to represent an image is to use the discrete wavelet transform (DWT). An advantage the DWT has over the Fourier transform and the discrete cosine transform is that it contains both frequency and location (spatial) information in its coefficients, whereas, both the Fourier and discrete cosine transforms only contain frequency information. The Matlab wavelet toolbox is used to perform the 2D wavelet transform on the eye images and the mother wavelet used is shown in figure 45.

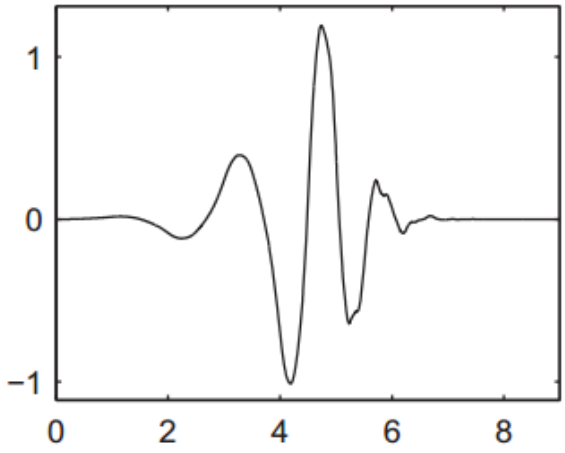



**Figure 45: db5 Mother Wavelet**

The db5 mother wavelet in figure 45 is able to store frequency and location information because its frequency components change throughout the function. The db5 wavelet was chosen as a compromise of compression and data retention. As can be seen in figure 45, the middle of the wavelet has a higher frequency than that of the wavelet near the beginning and end [8].

The one dimensional wavelet transform can be turned into a two dimensional wavelet transform by first applying the 1D transform to the columns of an image and then to the rows. This was done by taking advantage of Matlab functions wavedec2() and appcoef(). Once this is applied, the following image, figure 46, results.

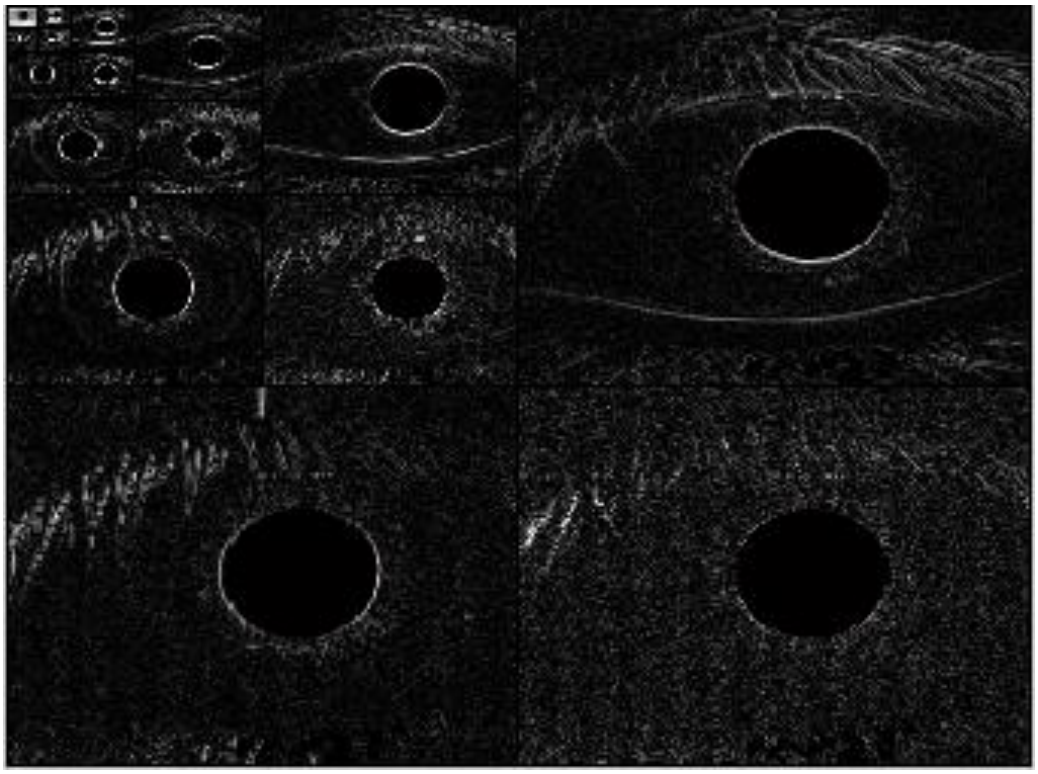


**Figure 46: 5th level decomposition of an eye image**

The image in figure 46 has 4 main components. First along the top are the representations of the one dimensional wavelet transform in the vertical direction (along columns). Along the left side are the horizontal representations of the one dimensional wavelet transform, and along the diagonal are the combinations of the individual one dimensional wavelet transformations. Lastly, the small image in the most upper left corner represents the 5 level average of the original image. The size of each level's decomposition is equal to

$$NumberPixels = \left( \| \frac{M}{2^d} \| * \| \frac{N}{2^d} \| \right) \quad \textbf{(14)}$$

In equation 14, M is equal to the number of rows in the original image, N is the number of columns in the original image, and d is the decomposition level. Also, the number of pixels is rounded to nearest whole value. For this thesis a fifth level decomposition is used, so the number of pixels in the decomposed image is 9x10, with the original image size 280 x 320 and d equal to 5. The reason why the fifth level decomposition is used is because it results in 90 coefficients, whereas the 4$^{th}$ level results in 360, too many for the neural network, and the 6$^{th}$ level results in 20 coefficients, too few for the neural network to converge quickly. The specific coefficients, 5$^{th}$ level detailed coefficients are highlighted in figure 47, which represents the detail of the low frequencies, and is what we are interested in, as a representation of the iris image.

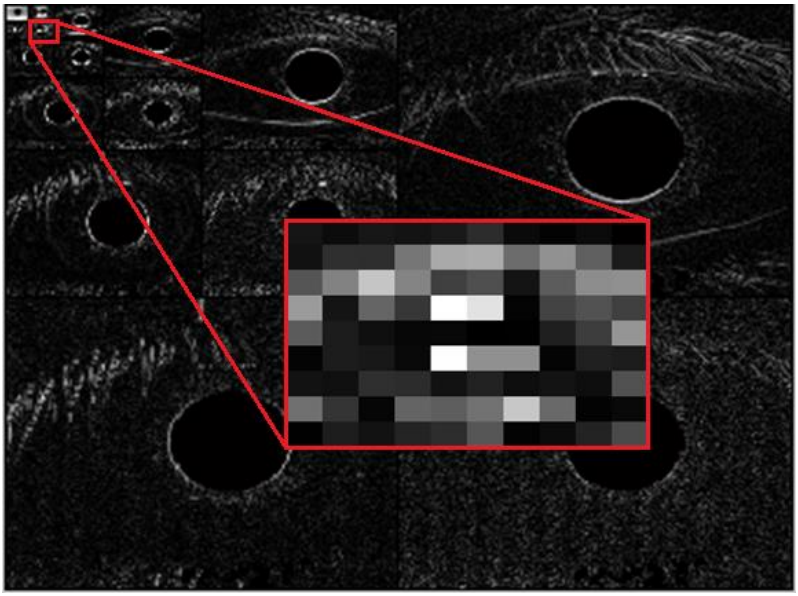


**Figure 47: 5$^{th}$ level decomposition coefficients used as inputs to the neural network**

WAVELET TRANSFORM RESULTS

When the 5 eyes are represented by 90 wavelet coefficients, the neural network classifies 90 percent of the test eyes input into the network. To examine the robustness of the wavelet coefficients as representations of the irises, noise is applied to the test images. The first and least avoidable noise is rotation. Rotation is tested with limits of plus and minus sixteen degrees and the results are shown in figure 48.

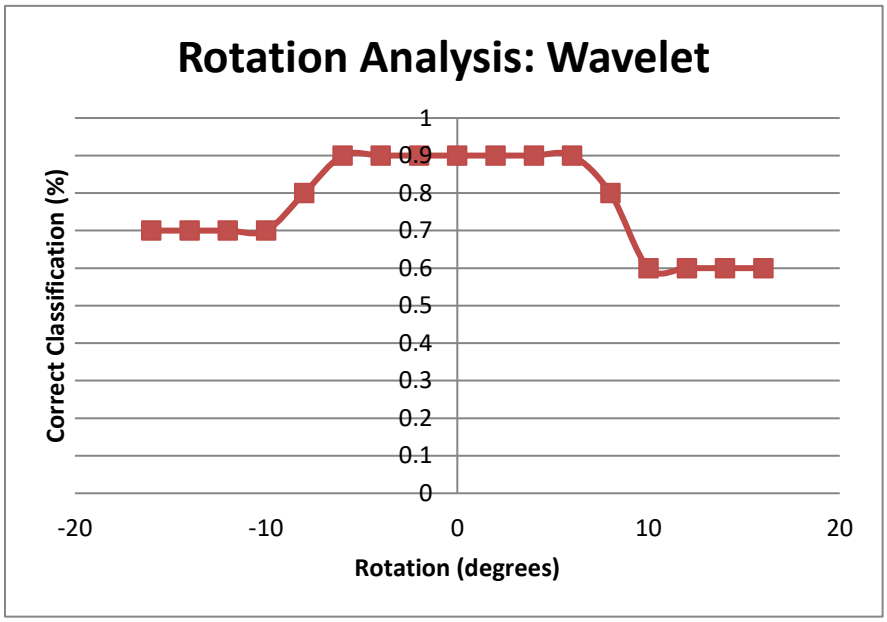



**Figure 48: Analysis on wavelet coefficients with added rotation distortion to test images**

Another way to display the plot of figure 48 is a confusion matrix, Table 8. As both the plot and confusion matrix show, the maximum degradation is 30 percent, which occurs at rotations above 10 degrees. If all the rotations are considered, the neural network trained with wavelet coefficients correctly classifies images at a rate of 76.47%.

**TABLE 8: Confusion matrix for wavelet coefficients with rotational distortion**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 34 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 28 | 0 | 2 | 0 | 4 |
| **3** | 11 | 0 | 23 | 0 | 0 | 0 |
| **4** | 0 | 5 | 0 | 26 | 0 | 3 |
| **5** | 1 | 7 | 0 | 4 | 19 | 3 |

The second type of noise that any camera will have is salt and pepper noise. Salt and pepper noise are randomly distributed white and black pixels throughout the image. Various amounts salt and pepper noise with SNR ranging from 5dB to 30dB, is applied to the test eye images and the results are summarized in the plot of figure 48.

The same test examined in figure 48 is explained in more detail with a confusion matrix, Table 9. The plot shows that any signal to noise ratio above 8dB has no effect on the correct classification of the eye images, whereas the confusion matrix shows what eyes become incorrectly classified. Based on the salt and pepper results of figure 49 and Table 9, the neural network trained with wavelet coefficients correctly classifies test images with salt and pepper noise at a rate of 87%.

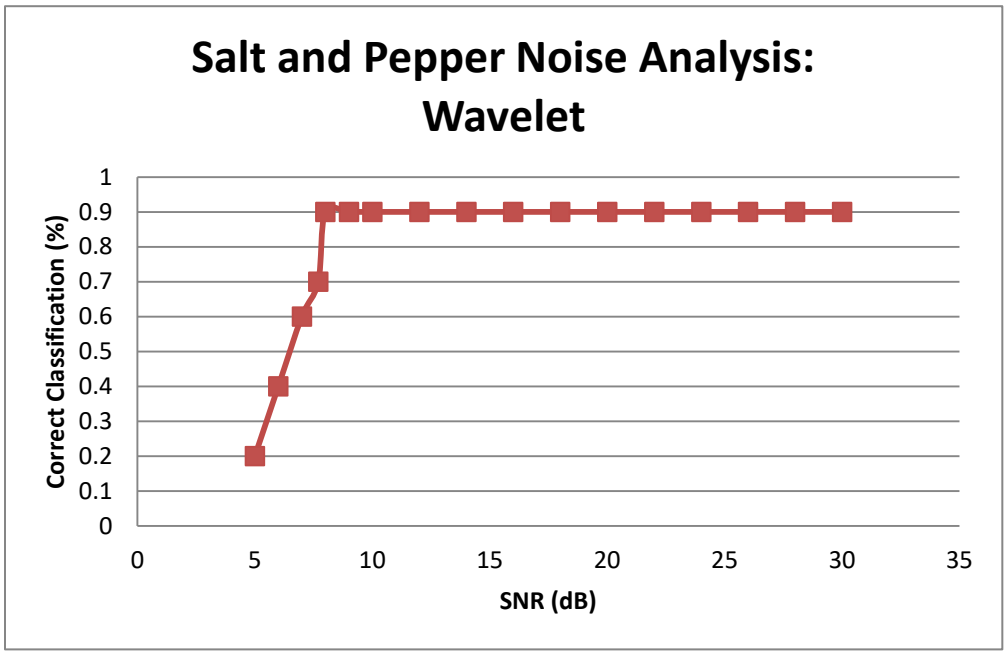



**Figure 49: Summary of Salt and Pepper noise analysis with wavelet coefficients**

**TABLE 9: Confusion matrix for salt and pepper test cases with SNR from 3 to 30dB**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 34 | 0 | 0 | 0 | 0 | 0 |
| **2** | 3 | 31 | 0 | 0 | 0 | 0 |
| **3** | 8 | 0 | 26 | 0 | 0 | 0 |
| **4** | 3 | 0 | 0 | 29 | 1 | 1 |
| **5** | 4 | 0 | 0 | 2 | 28 | 0 |

The third type of noise applied to the test images is Gaussian. After the neural network was trained with 5 images per eye, each eye has two different images which are used to test the neural network. These test images are convolved with a Gaussian kernel

of various standard deviations. The trained network is tested with wavelet coefficients from the noisy test eye images and summarized in figure 50.

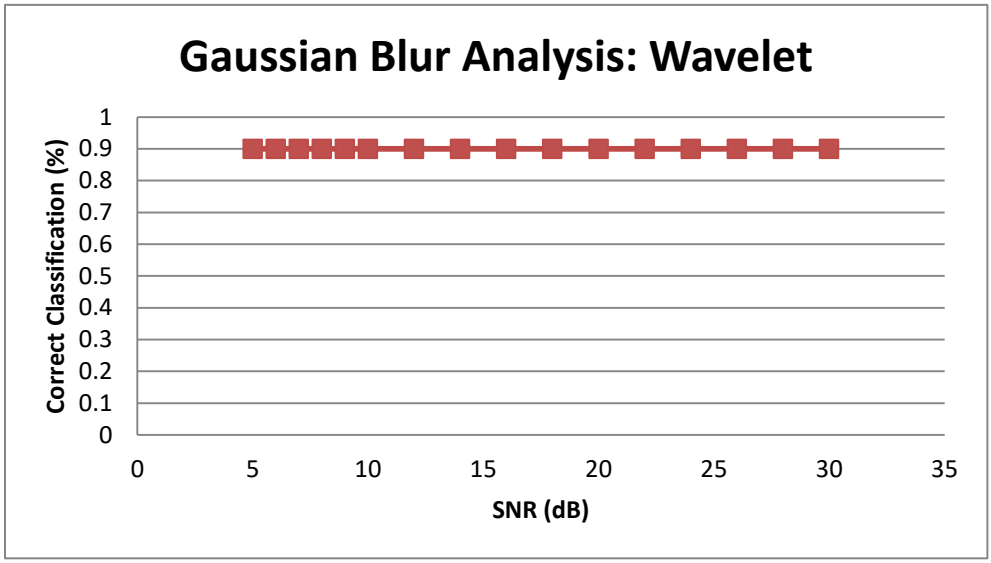



**Figure 50: Summary of Gaussian blur analysis with wavelet coefficients**

The same test examined in figure 50 is done but more details are saved with a confusion matrix, Table 10. Both the confusion matrix of table 10 and the plot of figure 50 show that no matter what the amount of Gaussian blur, the neural network correctly classifies at a 90 percent rate.

**TABLE 10: Confusion Matrix of Gaussian blur analysis with wavelet coefficients**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 34 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 34 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 34 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 34 | 0 | 0 |
| **5** | 0 | 0 | 0 | 17 | 17 | 0 |

## FOURIER AND EIGEN COEFFICIENTS

Eigen weights use spatial information of the iris to classify the eye, while Fourier transform coefficients use frequency information of the iris to classify the eye. By combining the two coefficient sets, as inputs to the neural network, the network will be able to use both frequency and spatial information to classify each eye.

The test will take all coefficients from each previous test. 121 Fourier transform coefficients and 50 Eigen weights, totaling 171 inputs to the neural network. All other variables remain the same, 5 different eyes, each with 5 different images of the same iris; will be used to train the network. Also the size of the neural network will remain at 15 first layer hidden neurons, 21 second layer hidden neurons and 11 third layer hidden neurons, which is the same as the Fourier transform Neural Network.

FOURIER AND EIGEN COEFFICIENTS RESULTS

The neural network correctly classified the test images 40 percent of the time. The three common types of noise, rotation, Gaussian, and salt and pepper, are applied to the test images and their effects on the robustness of the network are summarized in figures 51 through 53 and Tables 11 through 13.

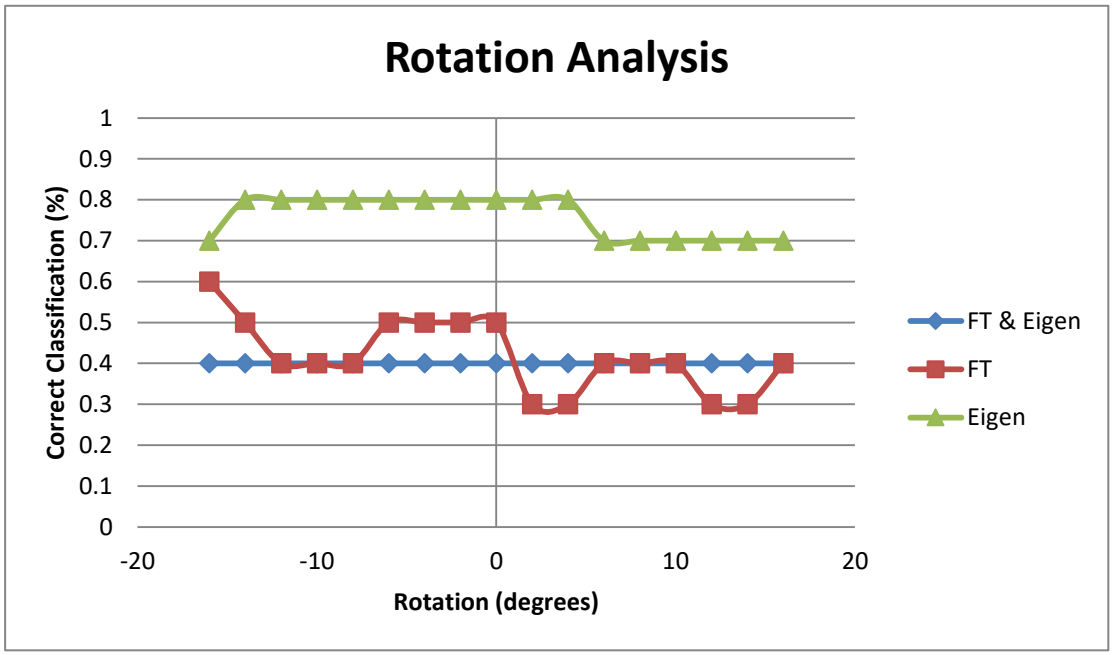



**Figure 51: Summary of rotation effect on Eigen + FT neural network**

**TABLE 11: Confusion matrix of rotational distortion of Eigen + FT neural network**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 17 | 0 | 0 | 17 | 0 | 0 |
| **2** | 0 | 17 | 0 | 0 | 17 | 0 |
| **3** | 0 | 0 | 17 | 0 | 0 | 17 |
| **4** | 0 | 17 | 0 | 17 | 0 | 0 |
| **5** | 17 | 17 | 0 | 0 | 0 | 0 |

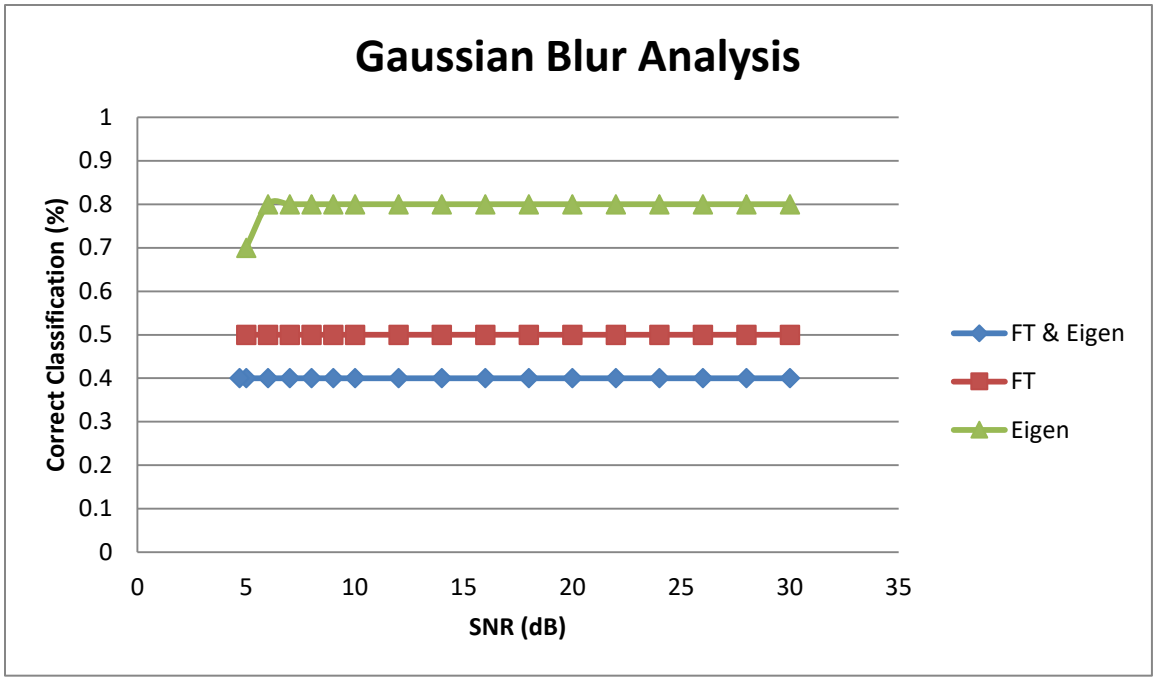



**Figure 52: Summary of Gaussian blur effect on Eigen +FT neural network**

**TABLE 12: Confusion matrix of Gaussian blur effect on Eigen + FT neural network**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 17 | 0 | 0 | 17 | 0 | 0 |
| **2** | 0 | 17 | 0 | 0 | 17 | 0 |
| **3** | 0 | 0 | 17 | 0 | 0 | 17 |
| **4** | 0 | 17 | 0 | 17 | 0 | 0 |
| **5** | 17 | 17 | 0 | 0 | 0 | 0 |

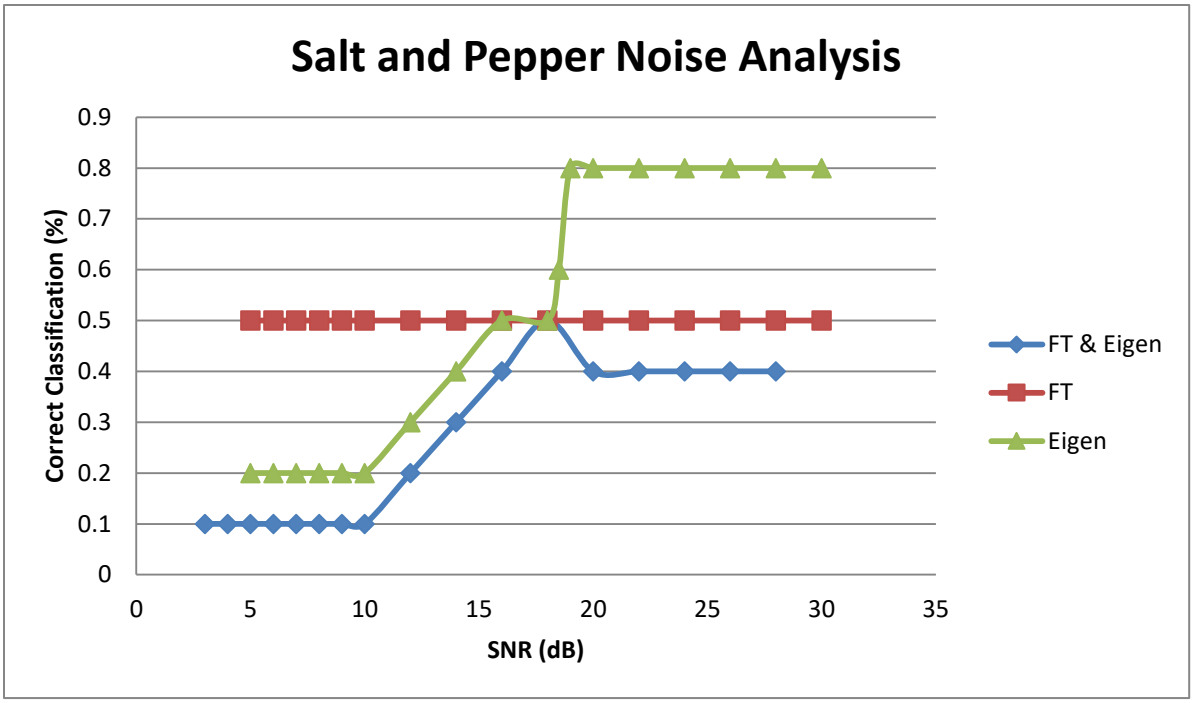



**Figure 53: Summary of Salt and Pepper noise effect on Eigen +FT neural network**

**TABLE 13: Confusion matrix of Salt and Pepper noise effect on Eigen + FT neural network**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 6 | 0 | 0 | 14 | 0 | 6 |
| **2** | 0 | 8 | 0 | 15 | 1 | 2 |
| **3** | 0 | 5 | 7 | 2 | 0 | 12 |
| **4** | 0 | 13 | 0 | 13 | 0 | 0 |
| **5** | 5 | 13 | 0 | 4 | 4 | 0 |

Based on the three noise and distortion test summaries, rotation (up to plus and minus 16 degrees) and Gaussian blur has no effect on the networks correct classification rate (always at 40%). The only noise to effect the network is when salt and pepper noise is added to the test images. The noise starts to affect the classification when the signal to noise ratio gets to 18dB. This SNR value is very similar to where the network that is only using Eigen weights as inputs is affected, shown in figure 39.

Overall, the combination of Fourier transform coefficients and Eigen space coefficients has worse classification than either of the individual classifier coefficients.

FOURIER AND WAVELET

Fourier transform coefficients hold frequency information of the eye images and the wavelet coefficients hold spatial and frequency information. The next tests combine 121 Fourier transform coefficients with 90 wavelet coefficients, figure 54. The first 121 coefficients are from the normalized Fourier transform and the next 90 are the normalized wavelet coefficients. These coefficients are normalized independently, see wavelet and Fourier sections for normalization procedures, and then concatenated

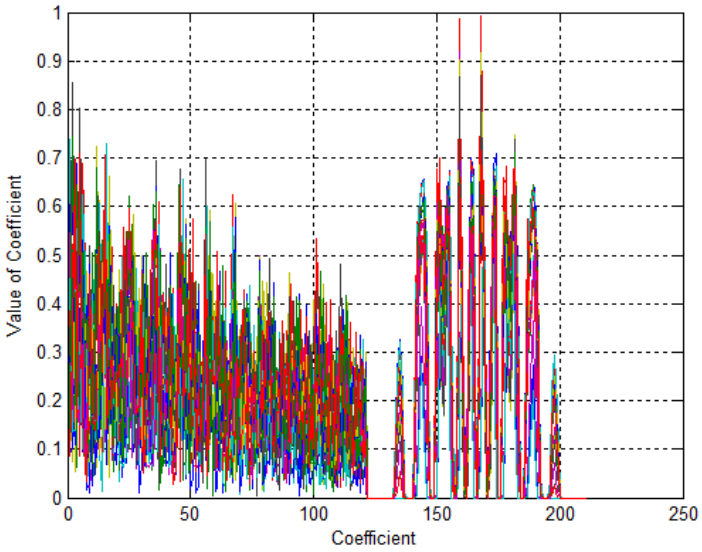



**Figure 54: graphical representation of combined FT and wavelet coefficients**

The neural network used for the combination of Fourier and Wavelet coefficients is [15, 21, 11] with $\alpha$=0.8, $\eta$=0.5, $\gamma$ = 0.3, and bias = 1 (hidden layers only).

FOURIER AND WAVELET RESULTS

The neural network correctly classified the test images 70 percent of the time. The three common types of noise, rotation, Gaussian, and salt and pepper, are applied to the test images and their effects on the robustness of the network are summarized in figures 55 through 57 and Tables 14 through 16.

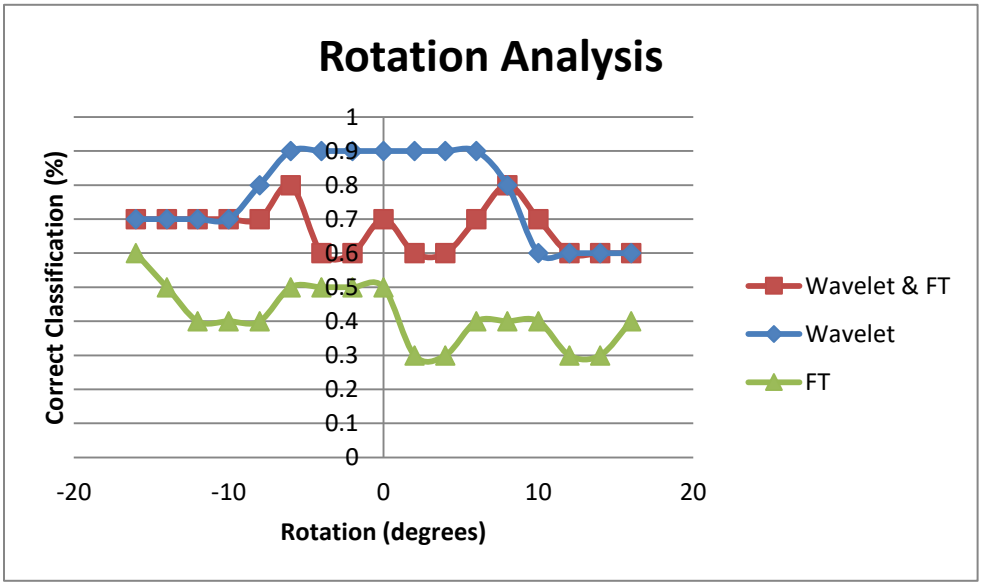



**Figure 55: Rotation analysis for FT and Wavelet coefficient combination**

**TABLE 14: Confusion matrix for FT and Wavelet coefficient combination with rotational distortion**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 17 | 0 | 17 | 0 | 0 | 0 |
| **2** | 0 | 34 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 34 | 0 | 0 | 0 |
| **4** | 0 | 16 | 0 | 13 | 0 | 5 |
| **5** | 6 | 0 | 8 | 1 | 19 | 0 |

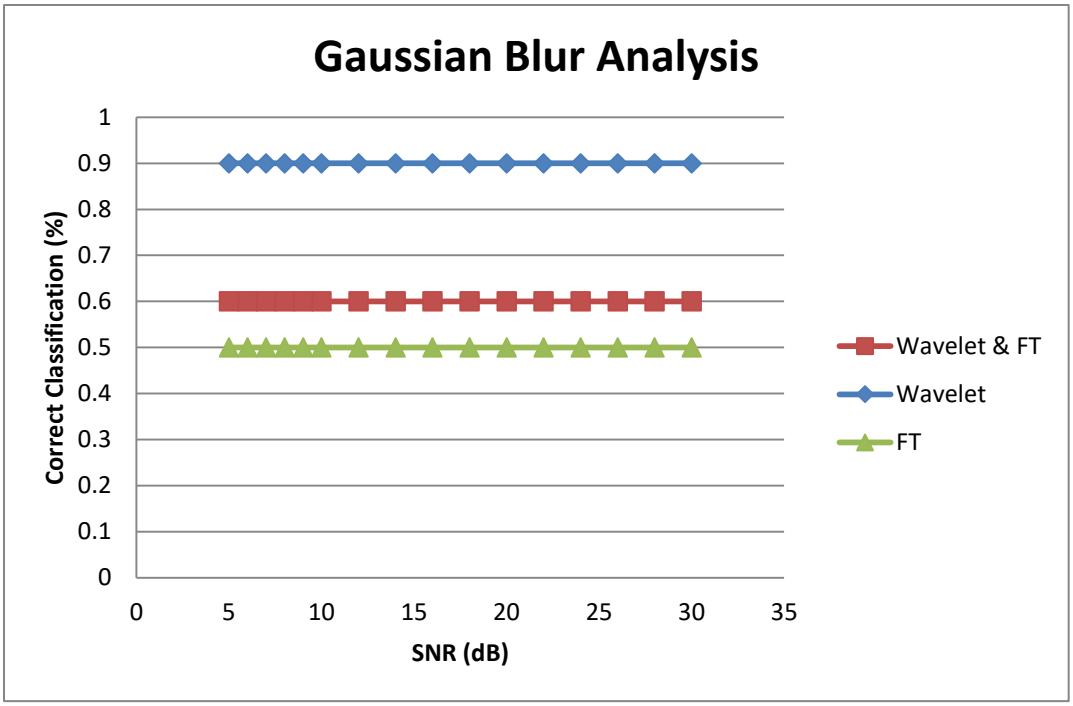



**Figure 56: Gaussian blur analysis for FT and Wavelet coefficient combination**

**TABLE 15: Confusion matrix for FT and Wavelet coefficient combination with Gaussian blur**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 17 | 0 | 17 | 0 | 0 | 0 |
| **2** | 0 | 34 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 34 | 0 | 0 | 0 |
| **4** | 0 | 17 | 0 | 17 | 0 | 0 |
| **5** | 34 | 0 | 0 | 0 | 0 | 0 |

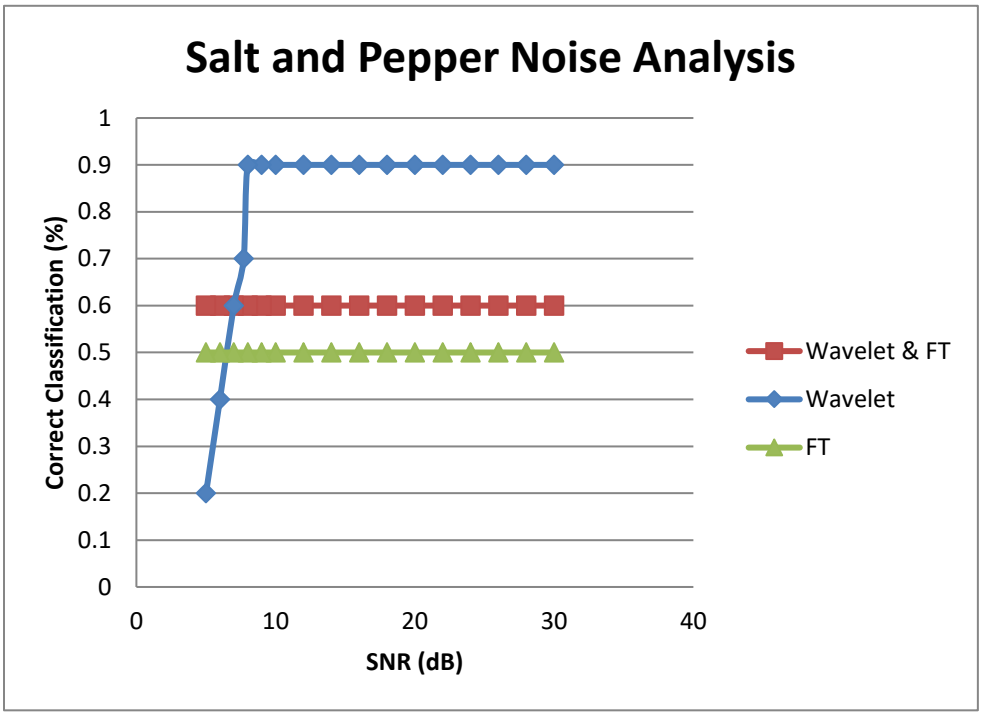



**Figure 57: Salt and Pepper noise analysis for FT and Wavelet coefficient combination**

**TABLE 16: Confusion matrix for FT and Wavelet coefficient combination with Salt and Pepper noise**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 17 | 0 | 17 | 0 | 0 | 0 |
| **2** | 0 | 34 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 34 | 0 | 0 | 0 |
| **4** | 0 | 17 | 0 | 17 | 0 | 0 |
| **5** | 34 | 0 | 0 | 0 | 0 | 0 |

Based on the three noise tests applied and summarized above, it can be seen that the combination of Fourier transform and wavelet coefficients almost always perform in-between the individual coefficient results. That means the wavelet outperforms the combination and the Fourier transform underperforms the combination.

Overall, the combination of Fourier and wavelet coefficients performs better than the Fourier coefficients individually but not as good as the wavelet coefficients individually.

DISCRETE COSINE TRANSFORM AND EIGEN

Eigen coefficients contain spatial information of the eye images and the discrete cosine transform coefficients contain frequency information. The discrete cosine transform outperformed the Fourier transform so this test will finalize the effect of adding Eigen coefficients (spatial information) to frequency information. The next tests combine 50 Eigen space coefficients with 121 discrete cosine transform coefficients. The first 50 coefficients are the normalized Eigen coefficients and the next 121 are normalized discrete cosine transform, figure 58.

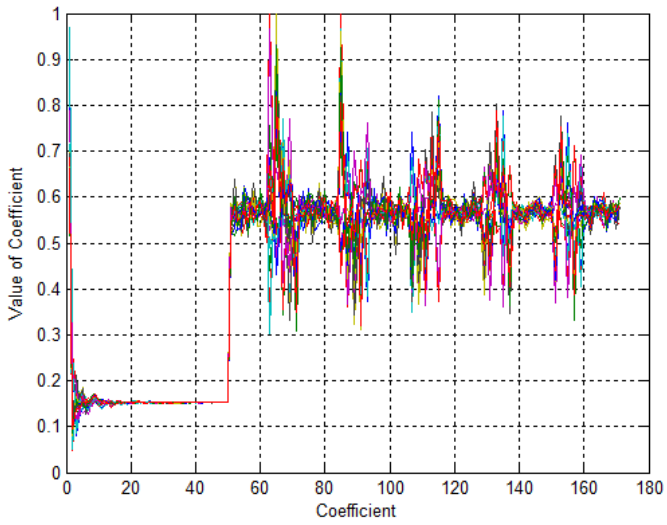



**Figure 58: Graphical representation of Eigen and DCT coefficients**

The neural network used for the combination of Eigen and discrete cosine coefficients is [15, 21, 11] with $\alpha$=0.8, $\eta$=0.5, $\gamma$ = 0.3, and bias = 1.

DISCRETE COSINE TRANSFORM AND EIGEN RESULTS

The neural network correctly classified the test images 90 percent of the time. The three common types of noise, rotation, Gaussian, and salt and pepper, are applied to the test images and their effects on the robustness of the neural network are summarized in figures 59 through 61 and Tables 17 through 19.

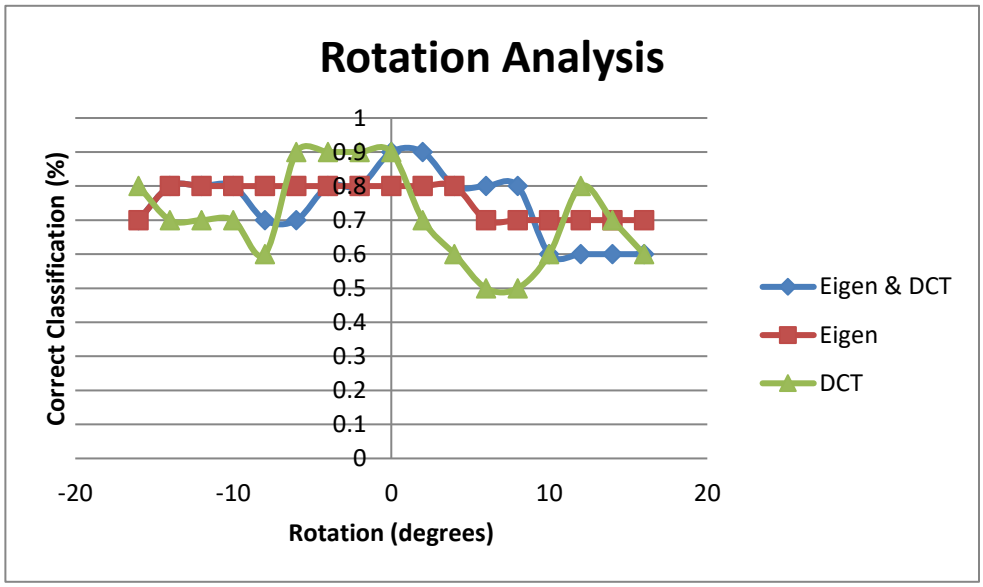



**Figure 59: Rotation analysis for Eigen and DCT coefficient combination**

**TABLE 17: Confusion matrix for Eigen and DCT coefficient combination with rotational distortion**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 34 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 8 | 0 | 23 | 0 | 3 |
| **3** | 8 | 0 | 25 | 0 | 1 | 0 |
| **4** | 0 | 0 | 0 | 34 | 0 | 0 |
| **5** | 6 | 0 | 2 | 0 | 26 | 0 |

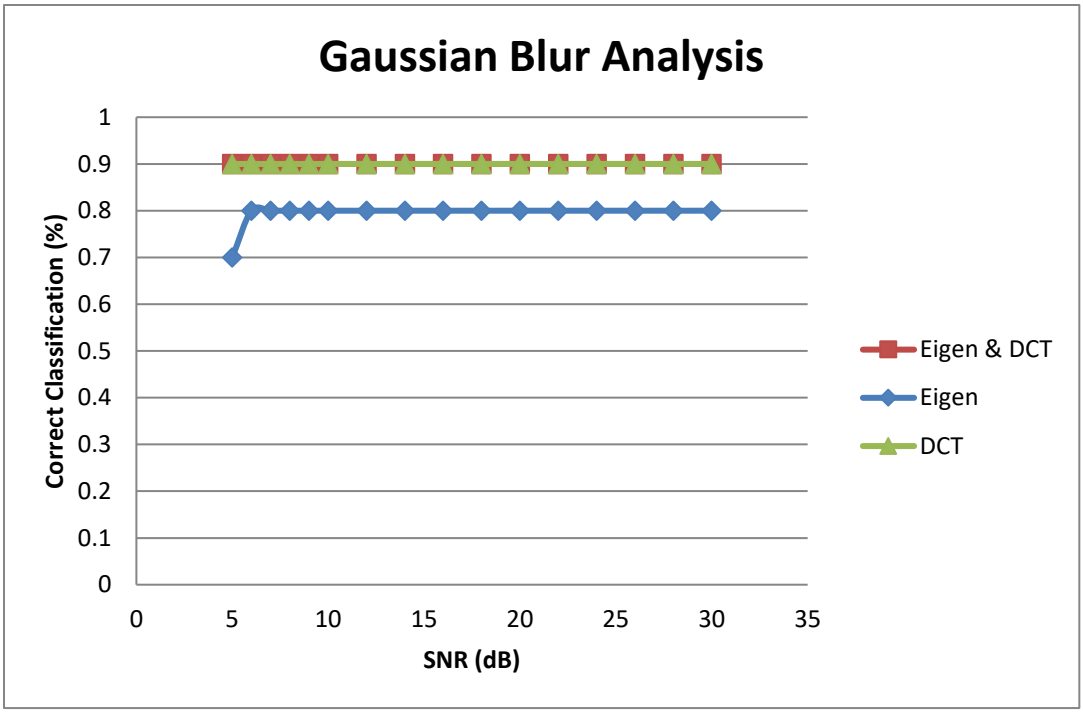



**Figure 60: Gaussian blur analysis for Eigen and DCT coefficient combination**

**TABLE 18: Confusion matrix for Eigen and DCT coefficient combination with Gaussian blur**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 32 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 8 | 0 | 24 | 0 | 0 |
| **3** | 6 | 0 | 23 | 0 | 3 | 0 |
| **4** | 0 | 0 | 0 | 32 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 32 | 0 |

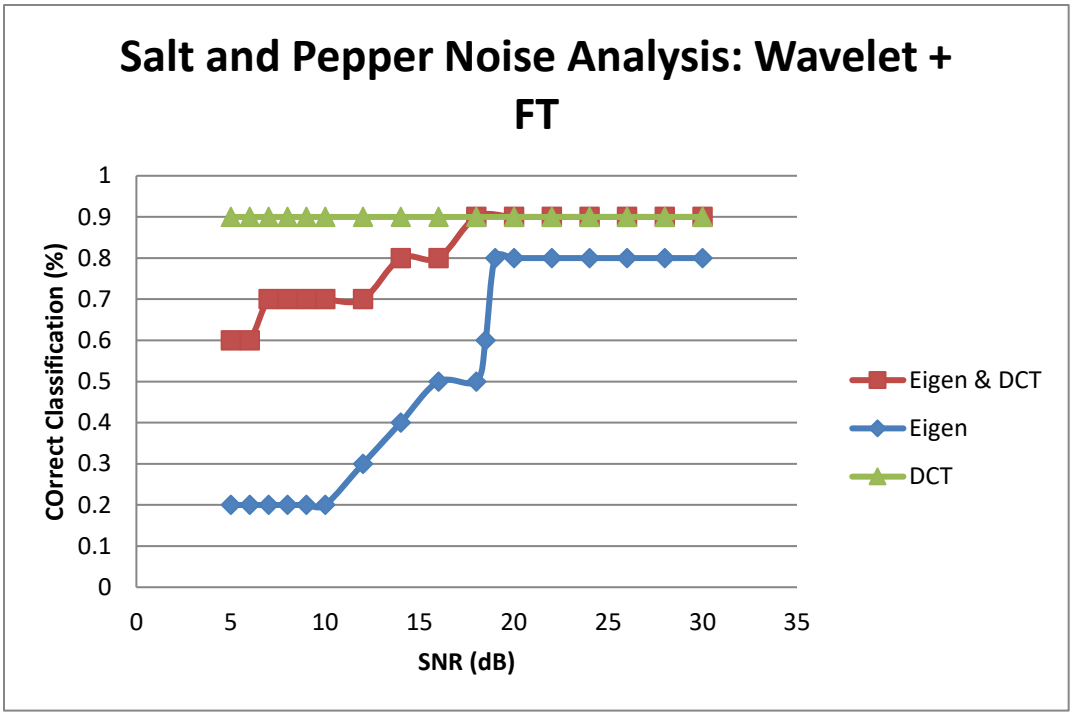



**Figure 61: Salt and Pepper noise analysis for Eigen and DCT coefficient combination**

**TABLE 19: Confusion matrix for Eigen and DCT coefficient combination with salt and pepper noise**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 32 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 8 | 0 | 24 | 0 | 0 |
| **3** | 6 | 0 | 23 | 0 | 3 | 0 |
| **4** | 0 | 0 | 0 | 32 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 32 | 0 |

Based on the noise tests applied and summarized above it can be seen that the combination of Eigen space coefficients and the discrete cosine transform coefficients is almost always worse than just DCT coefficients. There are only a few times where the combination of Eigen and DCT outperforms the DCT and this occurs when rotational distortion is applied to the images. This is likely an added property because the Eigen coefficients test well against rotation, as can be seen in figure 37.

Overall, the combination of the discrete cosine transform and Eigen coefficients do not perform better combined than individual DCT or Eigen coefficients.

WAVELET TRANSFORM AND DISCRETE COSINE TRANSFORM

Discrete Cosine transform coefficients hold frequency information of the eye images and the wavelet coefficients hold spatial and frequency information. The frequency information held in the two types of transforms is slightly different because the wavelet transform holds a more limited range of frequencies than the Fourier transform. This is due to the coefficients of the wavelet being taken from the 5$^{th}$ level representation. The next tests combine 121 DCT transform coefficients with 90 wavelet coefficients, figure 62. The first 121 coefficients are from the normalized discrete cosine transform and the next 90 are the normalized wavelet coefficients, totaling 211 coefficients.

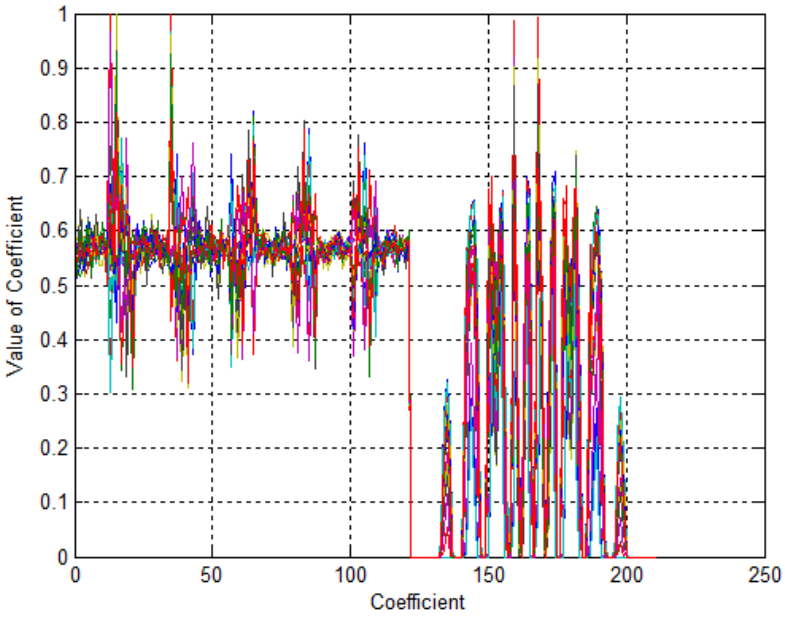



**Figure 62: Graphical representation of combined DCT and wavelet coefficients**

The neural network used for the combination of DCT and Wavelet coefficients is [15, 21, 11] with $\alpha$=0.8, $\eta$=0.5, $\gamma$ = 0.3, and bias = 1.

WAVELET TRANSFORM AND DISCRETE COSINE TRANSFORM RESULTS

The neural network correctly classified the test images 100 percent of the time. The three common types of noise, rotation, Gaussian, and salt and pepper, are applied to the test images and their effects on the robustness of the network are summarized in figures 63 through 65 and Tables 20 through 22.

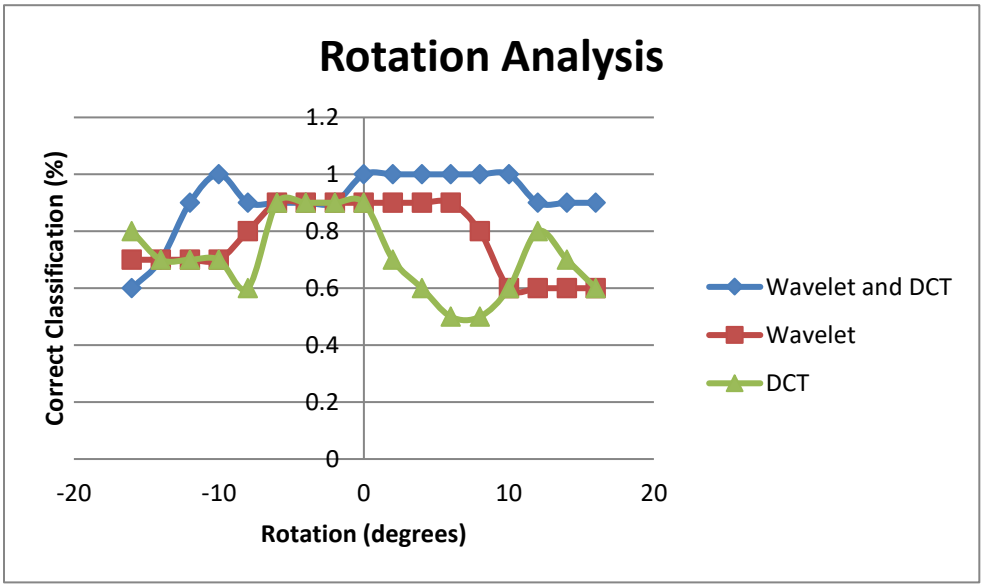



**Figure 63: Rotation distortion analysis for Wavelet and DCT coefficient combination**

**TABLE 20: Confusion matrix for wavelet and DCT coefficient combination with rotational distortion**

| | 1 | 2 | 3 | 4 | 5 | other |
|---|---|---|---|---|---|---|
| **1** | 34 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 29 | 0 | 0 | 0 | 5 |
| **3** | 3 | 0 | 31 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 34 | 0 | 0 |
| **5** | 0 | 0 | 0 | 7 | 27 | 0 |

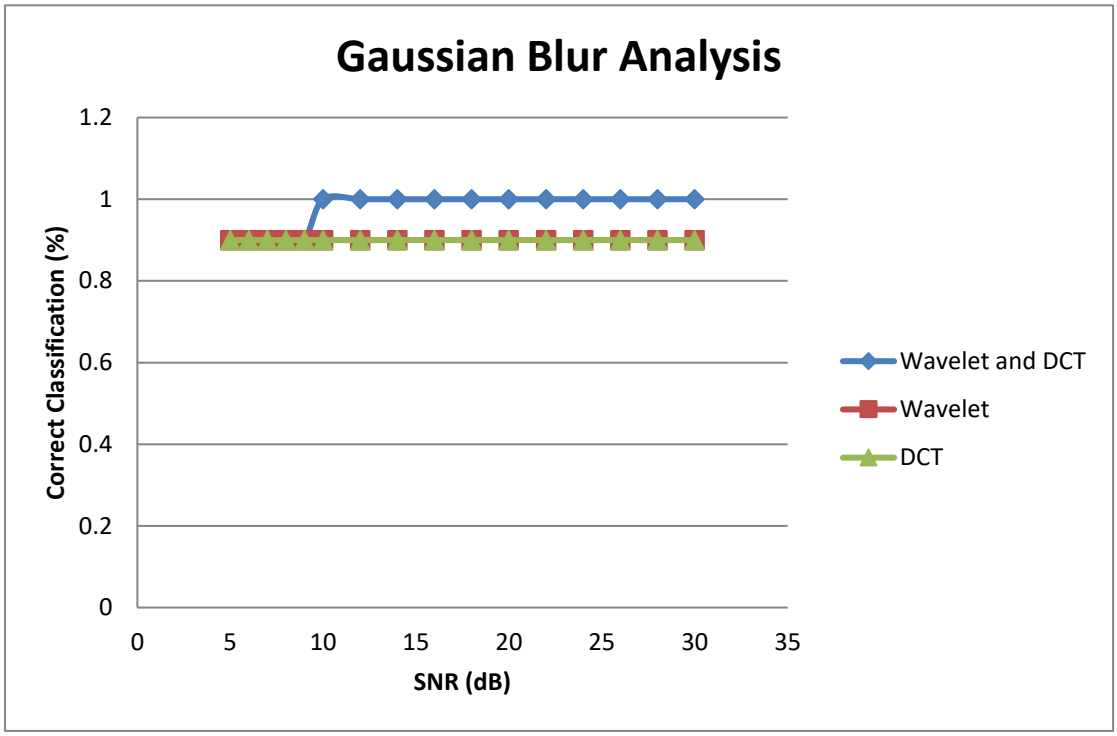



**Figure 64: Gaussian blur analysis for wavelet and DCT coefficient combination**

**TABLE 21: Confusion matrix for wavelet and DCT coefficient combination with Gaussian blur**

| | 1 | 2 | 3 | 4 | 5 | other |
|---|---|---|---|---|---|---|
| **1** | 34 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 34 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 34 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 34 | 0 | 0 |
| **5** | 5 | 0 | 0 | 0 | 29 | 0 |

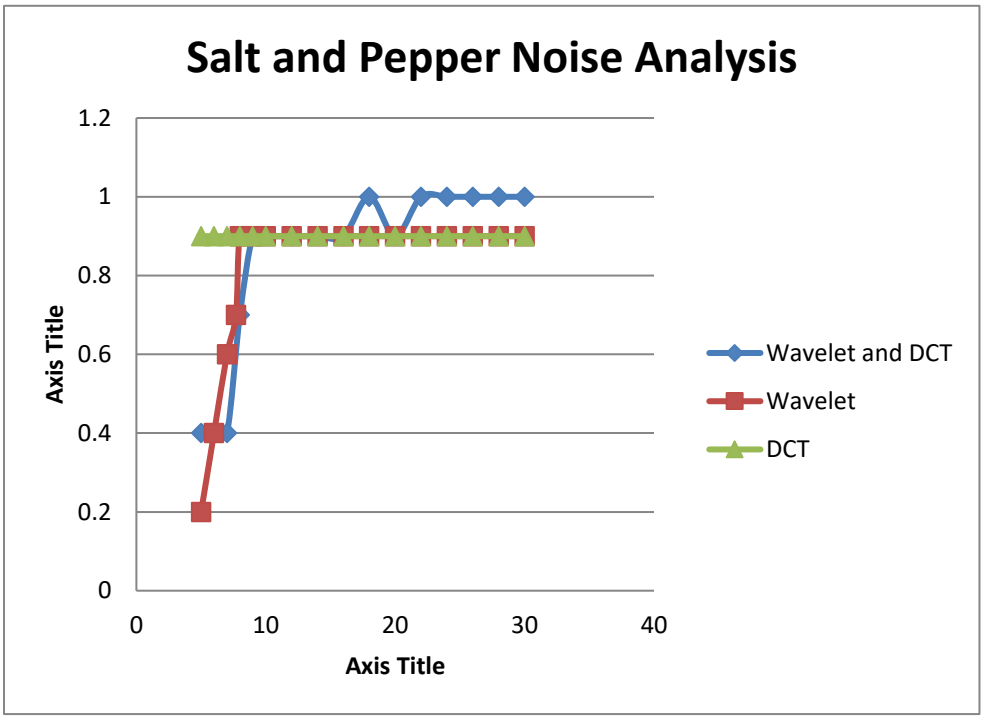



**Figure 65: Salt and Pepper noise analysis for wavelet and DCT coefficient combination**

**TABLE 22: Confusion matrix for wavelet and DCT coefficient combination with salt and pepper noise**

| | **1** | **2** | **3** | **4** | **5** | **other** |
|---|---|---|---|---|---|---|
| **1** | 32 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 26 | 6 | 0 | 0 | 0 |
| **3** | 3 | 0 | 29 | 0 | 0 | 0 |
| **4** | 1 | 0 | 0 | 28 | 3 | 0 |
| **5** | 14 | 0 | 0 | 0 | 18 | 0 |

Based on the three noise tests, the combination of wavelet transform coefficients and discrete cosine transform coefficients outperforms either individually for all but 1 point in the rotational distortion. As can be seen in figure 63 with the test images set to a -15 degree rotation the combination is outperformed by both of its components individually, but in all other points with the three different use case tests, the combination is as good or better than the individual transforms.

Overall, the combination of the discrete cosine transform and wavelet transform coefficients result in better classification than either of the two methods individually. This could be due to the information being held in DCT and wavelet coefficients complement each other. Meaning they hold different information and that the most significant DCT or wavelet coefficients are more important in classification than the less significant coefficients, since we are replacing the less significant coefficients of each transform with the more significant coefficients from the other transform.

**TABLE 23: Ranking of different input representations when no noise is applied**

| Method | Correct (%) |
|---|---|
| DCT+DWT | 100 |
| DWT | 90 |
| E+DCT | 90 |
| DCT | 90 |
| Eigen | 80 |
| FT+DWT | 70 |
| FT | 50 |
| FT+E | 40 |

**TABLE 24: Ranking input representations when rotational distortion is applied**

| Method | Correct (%) |
|---|---|
| DCT+DWT | 91 |
| DWT | 77 |
| Eigen | 76 |
| E+DCT | 75 |
| DCT | 71 |
| FT+DWT | 67 |
| FT | 42 |
| FT+E | 40 |

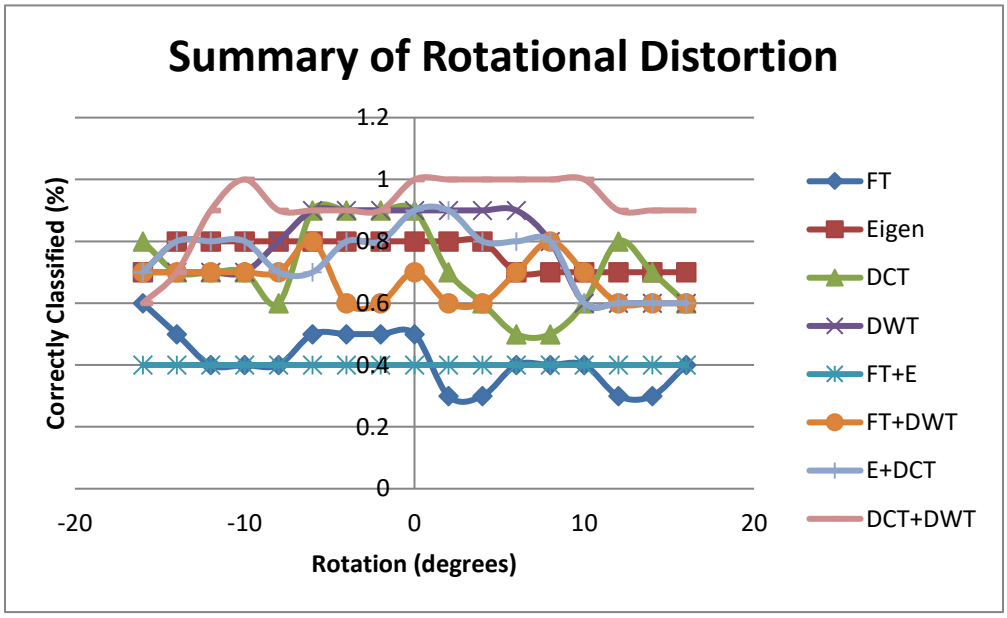



**Figure 66: Summary of all tests done with rotational distortion applied to test images**

Figure 66 shows the different combinations that have been examined and tested with rotational distortion. Out of all the different combinations it is clear that the best performer (with rotation) is discrete cosine transform plus the wavelet transform coefficients. Table 24 ranks from best performer to worst performer, with added rotational distortion. Table 23 is the baseline classifications with no noise or distortion.

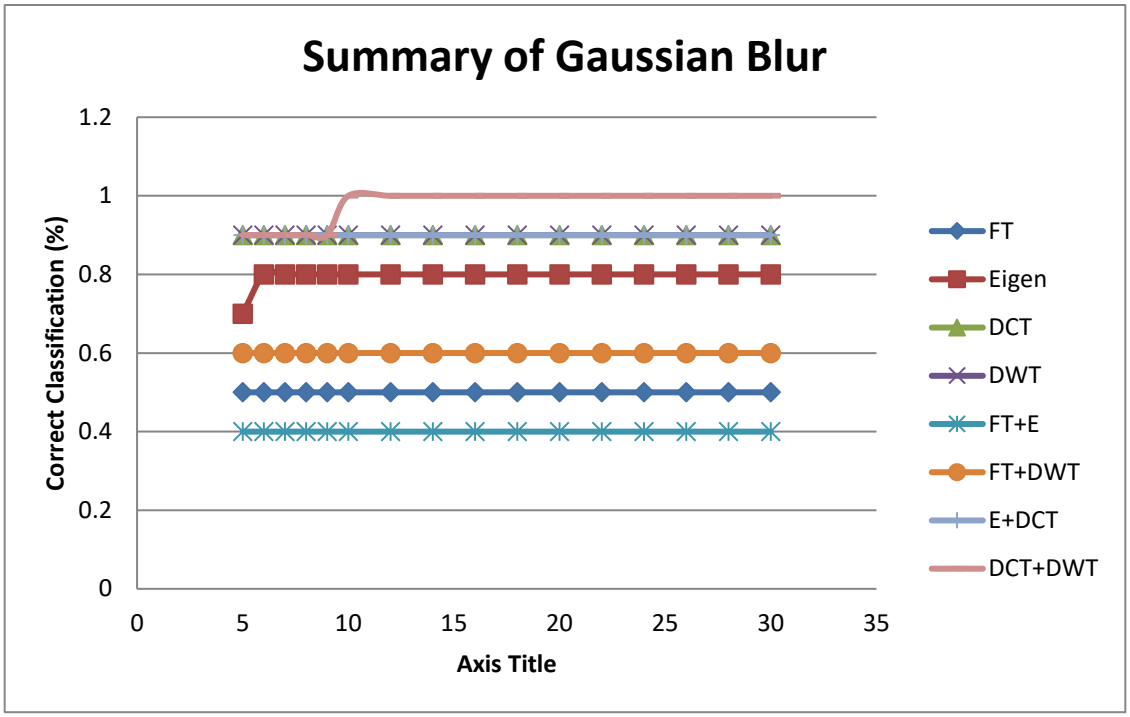



**Figure 67: Summary of all tests done with Gaussian blur applied to test images**

Figure 67 shows all the different combinations that have been examined and tested with Gaussian blur. Out of all the different combinations it is clear that the best performer when it comes to dealing with added Gaussian blur, is the combination of the discrete cosine transform and the wavelet transform coefficients. Table 25 ranks from best performer to worst performer, with added Gaussian blur.

**TABLE 25: Ranking of different input representations when Gaussian blur is applied**

| Method | Correct (%) |
|---|---|
| DCT+DWT | 97 |
| DWT | 90 |
| E+DCT | 90 |
| DCT | 90 |
| Eigen | 79 |
| FT+DWT | 60 |
| FT | 50 |
| FT+E | 40 |

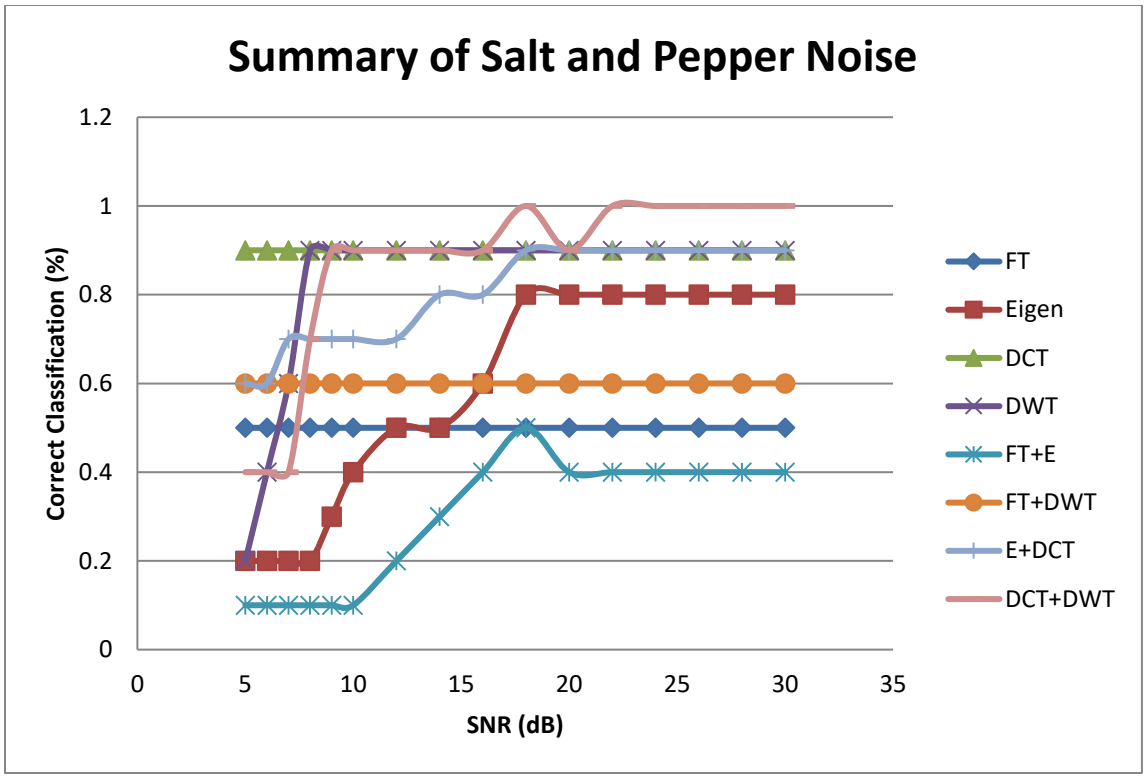



**Figure 68: Summary of all tests done with salt and pepper noise applied to test images**

Figure 68 shows all the different combinations that have been examined and tested with salt and pepper noise. It is a little harder to tell which method performs best. The combination of wavelet and DCT reach 100 classifications at low noise (high SNR) values but over the entire span the DCT has the highest overall performance. If the low SNR tests are ignored, which more resembles an image taken with a decent camera, the DCT and WVT combination performs best. Table 26 ranks from best to worst the performance of all methods over the entire SNR range of added salt and pepper noise.

**TABLE 26: Ranking of different input representations when salt and pepper noise is applied**

| Method | Correct (%) |
|---|---|
| DCT | 90 |
| DCT+DWT | 83 |
| DWT | 81 |
| E+DCT | 79 |
| FT+DWT | 60 |
| Eigen | 54 |
| FT | 50 |
| FT+E | 28 |

The specific values of the various coefficients were found by trial and error analysis for each of the different input transforms (Fourier, Eigen, DCT, and Wavelet), and the optimum results found are displayed below in Table 27. For the Fourier transform and Discrete Cosine transform 121 inputs are found to be the optimum number of inputs and also use the same size of Neural Network. The Eigen coefficients use a smaller number, 50, and therefore has a smaller Neural Network with fewer connections. The Wavelet transform uses 90 coefficients but only has a decrease in its optimum eta value.

**Table 27: Summary of Neural Networks used for each input methodology**

| | Neural Network Size | Inputs | Alpha ($\alpha$) | Eta ($\eta$) | Momentum ($\gamma$) |
|---|---|---|---|---|---|
| Fourier Transform | [15, 21, 11] | 121 | 0.8 | 0.6 | 0.3, MSE > 0.1<br>0.9, MSE $\leq$ 0.1 |
| Eigen | [10, 7, 10] | 50 | 0.3 | 0.5 | 0.3, MSE > 0.1<br>0.9, MSE $\leq$ 0.1 |
| Discrete Cosine Transform | [15, 21, 11] | 121 | 0.8 | 0.6 | 0.3, MSE > 0.1<br>0.9, MSE $\leq$ 0.1 |
| Wavelet Transform | [15, 21, 11] | 90 | 0.8 | 0.5 | 0.3, MSE > 0.1<br>0.9, MSE $\leq$ 0.1 |

FUTURE WORK

It was noticed that the logsig activation function, figure 69a, was used for all the neurons in the neural networks of this report. This could be a problem because all the inputs are normalized between zero and one, and that only falls onto the positive half of the activation function.

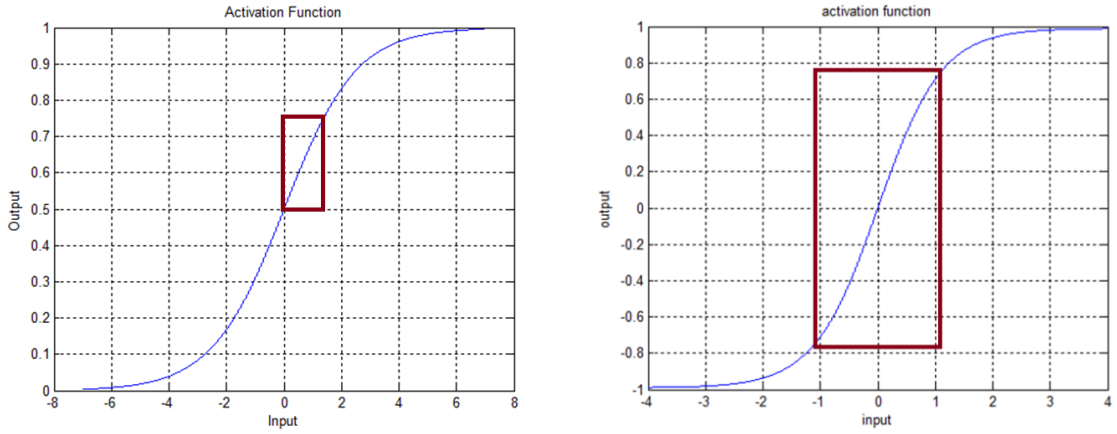



**Figure 69: (a) logsig activation function (b) tansig activation function**

To look into this possible problem all the activation functions were changed from logsig to tansig, figure 69b. The tansig activation function changes the output of each neuron from zero to one to negative one to one, enabling usage of almost the entire linear region of the function. The inputs were also normalized to range from negative one to one, utilizing more of the first layer's activation function. The neural network was then tested with discrete cosine transform (DCT) coefficients, which were normalized from -1 to 1, figure 70.

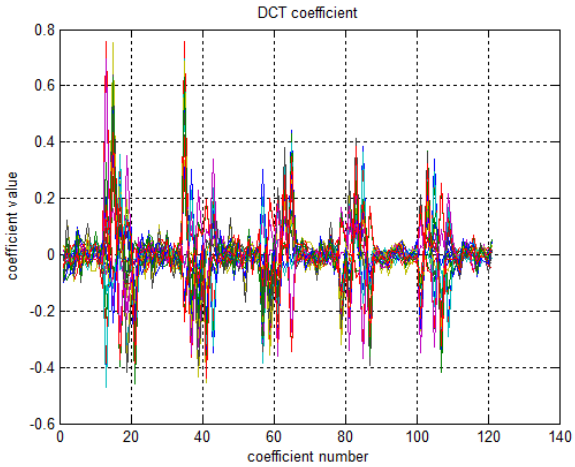



**Figure 70: DCT coefficients**

After ten different training runs with 121 coefficients and a neural network size of [15, 21, 11] the average correct classification between 5 eyes was 87%. Again, the network was tested with 50 Eigen inputs and a neural network size of [10, 7, 10] with all the same constants as with the network with the logsig activation function and after 10 different training runs the average classification percentage was 68. Although both of these results are slightly lower than the results from the original logsig activation function network, the time it took for the neural network to converge to a mean square error of 0.01 was significantly lower, summarized in Table 28.

**TABLE 28: Time of Network Convergence**

| | Logsig (0,1) | Tansig (-1,1) |
|---|---|---|
| DCT | 1928 sec | 746 sec |
| Eigen | 3945 sec | 1286 sec |

The numbers in Table 28 are the time, in seconds, that it took the neural network to converge to a mean square error of 0.01. Each test was run 10 times and the average time of convergence was taken. It can be seen that the logsig activation function is more than 2.5 times slower than that of the network that uses the tansig activation function. Overall the tansig activation function did not perform more accurately than the logsig activation function, but it did allow the neural network to train faster, which is a significant advantage as the number of eyes being classified increases.

To further the research into using neural networks as a classifier, more eyes can be added to check for a larger number of classified eyes. This will check the scalability of each technique. Based on the findings of the performed experiments, the most promising technique is the combination of discrete cosine transform coefficients and

wavelet coefficients. Also, further research into whether to use a tansig or logsig activation function should be performed.

To better the detection rate, additional boundary sets, instead of just three, could be found to allow for a more accurate average when finding the final pupil boundary and center. The biggest problem with the pupil detection algorithm is the interference with eyelashes. Preprocessing can be applied to the eye images to limit the interference from eyelashes by removing them entirely from the image [12]. Another improvement that could be added to future pupil detection algorithms, is to limit the range of the pupil radius. From the 700 different images the average pupil radius is 44.9 pixels, with the largest being 63 and the smallest being 33 pixels. If the algorithm knew this prior to the pupil detection it could limit its search locations, therefore speeding up the time needed to detect the pupil. This would decrease the time needed in order to run the boundary detection phase. This range limitation is not applied in this segmentation algorithm but would be implemented if a new set of iris images were to be tested.

Based on the conclusion of this thesis, that the best transform for correct classification were the Discrete Cosine Transform, the Discrete Wavelet Transform, and their combination, more tests are performed with these inputs.

First the number of eyes are expanded from 5 to 8, since this is the largest number of possible outputs the network can support with 3 outputs using binary encoding. The test images are expanded to 5 (non-training) images of each eye. These 5 test eyes have Gaussian (SNR from 5 to 30dB, 26 images), rotational (-16 to 16 degrees, 33 images), and salt and pepper noise (SNR from 5 to 30dB, 26 images) applied and then input into the trained neural network. This results in 3400 tests per each trained artificial neural

network. To test the robustness of the Discrete Cosine coefficients the neural network was trained 8 times to get 8 different sets of weights and the 3400 test images were applied to each network and results are summarized in the confusion matrix seen in Tables 29 through 31.

$$\frac{8\ eyes}{ANN} * \frac{5\ test\ image}{eye} * \frac{85\ tests}{test\ image} = 3400\ tests\ per\ ANN$$

**TABLE 29: Confusion Matrix of Discrete Cosine Transform Results**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 3056 | 1 | 1 | 11 | 21 | 0 | 0 | 310 |
| 2 | 48 | 2550 | 11 | 287 | 18 | 241 | 4 | 241 |
| 3 | 180 | 22 | 3070 | 23 | 17 | 50 | 2 | 36 |
| 4 | 21 | 2 | 85 | 3235 | 26 | 2 | 3 | 26 |
| 5 | 25 | 4 | 5 | 159 | 3025 | 3 | 171 | 8 |
| 6 | 10 | 473 | 16 | 184 | 86 | 2461 | 121 | 49 |
| 7 | 10 | 5 | 15 | 165 | 211 | 180 | 2795 | 19 |
| 8 | 20 | 3 | 1 | 77 | 122 | 9 | 72 | 3096 |

From the confusion matrix above, Table 29, we can calculate the correct classification percent by adding up all the values along the diagonal (test images that were correctly classified) and dividing by the total number of tests run (sum of all numbers in matrix).

$$100(\frac{3056 + 2550 + 3070 + 3235 + 3025 + 2461 + 2795 + 3096}{27200}) = 85.62\%$$

**TABLE 30: Confusion Matrix of Discrete Wavelet Transform Results**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 3090 | 2 | 8 | 0 | 1 | 0 | 1 | 298 |
| 2 | 43 | 3112 | 6 | 10 | 6 | 11 | 0 | 212 |
| 3 | 234 | 154 | 2974 | 0 | 10 | 0 | 10 | 18 |
| 4 | 0 | 10 | 89 | 3181 | 10 | 56 | 0 | 54 |
| 5 | 175 | 3 | 1 | 144 | 3016 | 0 | 39 | 22 |
| 6 | 0 | 509 | 5 | 53 | 108 | 2669 | 44 | 12 |
| 7 | 5 | 35 | 27 | 24 | 111 | 475 | 2714 | 9 |
| 8 | 0 | 2 | 0 | 6 | 5 | 0 | 233 | 3154 |

From the confusion matrix above we can calculate the correct classification percent by adding up all the values along the diagonal (test images that were correctly classified) and dividing by the total number of tests run (sum of all numbers in matrix).

$$100\left(\frac{3090 + 3112 + 2974 + 3181 + 3016 + 2669 + 2714 + 3154}{27200}\right) = 87.90\%$$

**TABLE 31: Confusion Matrix of Discrete Wavelet and Cosine Transform Results**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 3078 | 2 | 5 | 0 | 11 | 30 | 16 | 258 |
| 2 | 44 | 3076 | 40 | 7 | 3 | 62 | 23 | 145 |
| 3 | 31 | 82 | 3252 | 3 | 2 | 12 | 8 | 10 |
| 4 | 0 | 8 | 104 | 3192 | 10 | 40 | 8 | 38 |
| 5 | 59 | 2 | 2 | 191 | 2910 | 132 | 102 | 2 |
| 6 | 3 | 491 | 7 | 92 | 30 | 2687 | 84 | 6 |
| 7 | 2 | 8 | 17 | 45 | 58 | 515 | 2753 | 2 |
| 8 | 9 | 19 | 3 | 22 | 32 | 27 | 221 | 3067 |

From the confusion matrix above we can calculate the correct classification percent by adding up all the values along the diagonal (test images that were correctly classified) and dividing by the total number of tests run (sum of all numbers in matrix).

$$100\left(\frac{3078 + 3076 + 3252 + 3192 + 2910 + 2687 + 2753 + 3067}{27200}\right) = 88.29\%$$

It is important to note here that the DCT+WVT coefficients were created by taking half of each (first half) individual DCT and WVT coefficient set.

BIBLIOGRAPHY


1. Abiyev, Rahib, and Koray Altunkaya. "Personal iris recognition using neural network." *International Journal of Security and its Applications* 2.2 (2008
2. Baqar, M.; Azhar, S.; Iqbal, Z.; Shakeel, I.; Ahmed, L.; Moinuddin, M., "Efficient iris recognition system based on dual boundary detection using robust variable learning rate Multilayer Feed Forward neural network," *Information Assurance and Security (IAS), 2011 7th International Conference on* , vol., no., pp.326,330, 5-8 Dec. 2011 doi: 10.1109/ISIAS.2011.6122841
3. Boles, W.W.; Boashash, B., "A human identification technique using images of the iris and wavelet transform," *Signal Processing, IEEE Transactions on* , vol.46, no.4, pp.1185,1188, Apr 1998
   doi: 10.1109/78.668573
4. C. Zhang and T. H. Keung Kwan, “Parameter effects on convergence speed and generalization capability of backpropagation algorithm,” International Journal of Electronics, vol. 74, Jan. 1993, pp.35–46, doi: 10.1080/00207219308925810.
5. CASIA Iris Image Database, 2007, Chinese Academy of Sciences. Version 3. 60 subjects, 2400 images.
6. El-Bakry, H.M.; , "Fast iris detection for personal identification using modular neural networks," *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on* , vol.3, no., pp.581-584 vol. 2, 6-9 May 2001
   doi: 10.1109/ISCAS.2001.921377

7. Farouk, R.M.; Kumar, R.; Riad, K.A.; , "Iris matching using multi-dimensional artificial neural network,"*Computer Vision, IET* , vol.5, no.3, pp.178-184, May 2011 doi: 10.1049/iet-cvi.2010.0133
8. Gonzalez, Rafael C., and Richard E. Woods. Digital image processing. Upper Saddle River, N.J: Prentice Hall, 2008. Print.
9. Labati, R.D.; Piuri, V.; Scotti, F.; , "Neural-based iterative approach for iris detection in iris recognition systems," *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on* , vol., no., pp.1-6, 8-10 July 2009
doi: 10.1109/CISDA.2009.5356533
10. Masek, Libor. "Recognition of human iris patterns for biometric identification." *M. Thesis, The University of Western Australia* 3 (2003).
11. Miao Zhenjiang; Yuan Baozong; , "An extended BAM neural network model," *Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on* , vol.3, no., pp. 2682- 2685 vol.3, 25-29 Oct. 1993 doi: 10.1109/IJCNN.1993.714276
12. Miyazawa, Kazuyuki; Ito, Koichi; Aoki Takafumi; , “A Phase-Based Iris Recognition Algorithm,” Tohoku University, Japan, pp 356-365 2005
13. Vuillemin, J.E.; , "Fast linear Hough transform," *Application Specific Array Processors, 1994. Proceedings. International Conference on* , vol., no., pp.1-9, 22-24 Aug 1994
doi: 10.1109/ASAP.1994.331821

14. Wildes, R.P.; , "Iris recognition: an emerging biometric technology," *Proceedings of the IEEE* , vol.85, no.9, pp.1348-1363, Sep 1997
doi: 10.1109/5.628669

15. Xing Chen; Ling Lu; Yang Gao; , "A new concentric circle detection method based on Hough transform,"*Computer Science & Education (ICCSE), 2012 7th International Conference on* , vol., no., pp.753-758, 14-17 July 2012
doi: 10.1109/ICCSE.2012.6295182

16. Yong Li; Yang Fu; Hui Li; Si-Wen Zhang, "The Improved Training Algorithm of Back Propagation Neural Network with Self-adaptive Learning Rate," *Computational Intelligence and Natural Computing, 2009. CINC '09. International Conference on* , vol.1, no., pp.73,76, 6-7 June 2009
doi: 10.1109/CINC.2009.111

17. Yong Zhu; Tieniu Tan; Yunhong Wang, "Biometric personal identification based on iris patterns," *Pattern Recognition, 2000. Proceedings. 15th International Conference on* , vol.2, no., pp.801,804 vol.2, 2000
doi: 10.1109/ICPR.2000.906197

18. Zhong Bo Zhang; Si Liang Ma; Ping Zuo; Jie Ma; , "Fast Iris Detection and Localization Algorithm Based on AdaBoost Algorithm and Neural Networks," *Neural Networks and Brain, 2005. ICNN&B '05. International Conference on* , vol.2, no., pp.1085-1088, 13-15 Oct. 2005
doi: 10.1109/ICNNB.2005.1614806

19. Sbig. “Operating Manual for STF-8300 Series Cameras.”, 8 Dec. 2011, p. 11., www2.astro.puc.cl/ObsUC/images/d/d6/STF-8300_Manual_12.pdf

APPENDIX A: MATLAB CODE

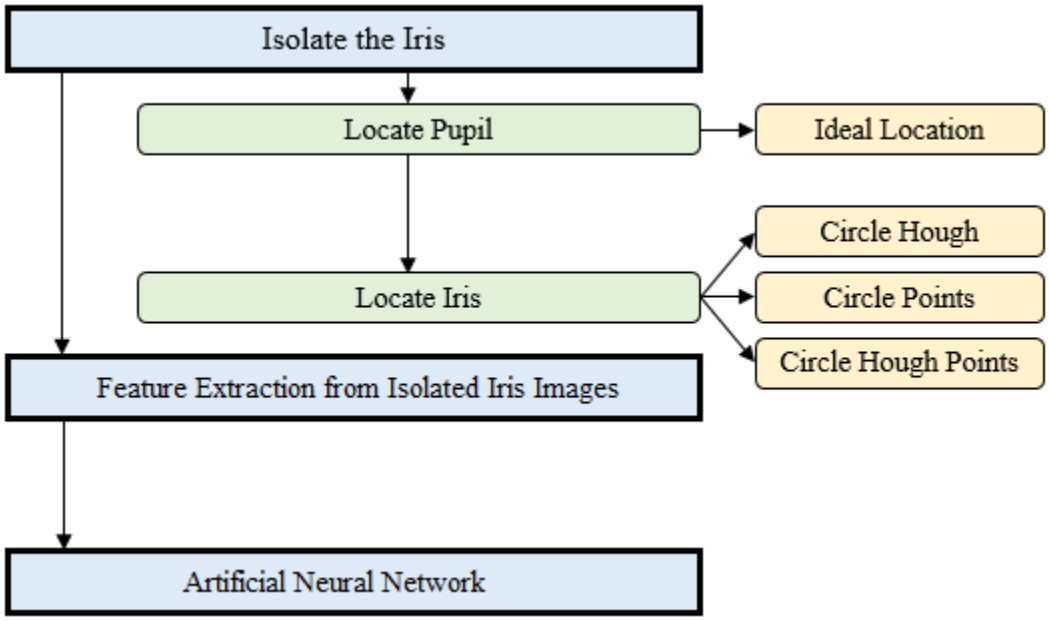



**Figure 71: Flow Diagram of Matlab code**

## ISOLATE THE IRIS

```
clear all; close all; clc;
tic
BLOCK =10;
number = 0;
```

Add the location of the original iris images to the search path

```
addpath('C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris
Biometric_for_writeup\Finals Week Combination\Original_Images');

for image_num = 1
```

---

```
   number = number+1;
% read in image(s) and find its size
   iris_bw = strcat('(', num2str(image_num), '.bmp');
   iris_image = imread(iris_bw);
   [M,N] = size(iris_image);

% detect pupil boundary
   [m_pupil_center, n_pupil_center, pupil_radius, M_pupil_locations,...
      N_pupil_locations,iris_image_circled_pupil] ...
      = locate_pupil(iris_image, BLOCK);

% detect iris boundary
   [edge_image,thresh] = edge(iris_image,'canny',[],3.5);
   [iris_image_circled_iris,iris_radius, m_iris, n_iris] = ...
       locate_iris(iris_image,edge_image,m_pupil_center, n_pupil_center,pupil_radius);

% create iris mask
   iris_mask = zeros(M,N);

   % label iris pixels as 1
      for i = 1:length(m_iris)
         iris_mask(m_iris(i),n_iris(i)) = iris_image(m_iris(i),n_iris(i));
      end

   % label pupil pixels as 0
      for i = 1:length(M_pupil_locations)
         iris_mask(M_pupil_locations(i),N_pupil_locations(i)) = 1-
iris_image(M_pupil_locations(i),N_pupil_locations(i));
      end

   % create mask (iris locations = 1)
      iris_mask = double(iris_mask>0);


% multiply the mask with the iris_image to isolate the iris
      isolated_iris = uint8(double(iris_image).*double(iris_mask));
      iris_with_lashes = isolated_iris;

%  center the iris
   center_shift_m = 140-m_pupil_center;
```

```
    center_shift_n = 160-n_pupil_center;

        iris_center = zeros(size(iris_with_lashes));
        mask_center = zeros(size(iris_with_lashes));
        for i = 1:280
            for j = 1:320
                if iris_mask(i,j) == 1
                    iris_center(i+center_shift_m ,j+center_shift_n ) = iris_with_lashes(i,j);
                    mask_center(i+center_shift_m ,j+center_shift_n ) = 1;
                end
            end
        end
        iris_center = uint8(iris_center);

% OUTPUTS
        isolated(number).lashes = iris_center;
        isolated(number).mask = mask_center;
```

Makes more images for testing purposes (rotations)

```
% rotate from 1 to 30 and then from 330 to 359 to add 60 test images
angle = [round(linspace(2,30,10)), round(linspace(330,359,3))];

i = 0;
for angle = angle
    i =i+1;
    % rotate without lashes
        iris_with_lashes_rotate(:,:,i) = imrotate(iris_with_lashes,angle,'bilinear','crop');
        mask_with_lashes_rotate(:,:,i) = imrotate(iris_mask,angle,'bilinear','crop');
        FT_iris_with_lashes_rotate(:,:,i)  = fourier(iris_with_lashes_rotate(:,:,i));
        FT_mask_with_lashes_rotate(:,:,i) = fourier(mask_with_lashes_rotate(:,:,i));
        FT_iris_with_lashes_rotate_minus_mask(:,:,i)  = FT_iris_with_lashes_rotate(:,:,i) -
FT_mask_with_lashes_rotate(:,:,i) ;

    % rotate with lashes
        iris_without_lashes_rotate(:,:,i) = imrotate(isolated_iris,angle,'bilinear','crop');
        mask_without_lashes_rotate(:,:,i) = imrotate(total_mask,angle,'bilinear','crop');
        FT_iris_without_lashes_rotate(:,:,i)  = fourier(iris_without_lashes_rotate(:,:,i));
        FT_mask_without_lashes_rotate(:,:,i) = fourier(mask_without_lashes_rotate(:,:,i));
        FT_iris_without_lashes_rotate_minus_mask(:,:,i)  = FT_iris_without_lashes_rotate(:,:,i) -
FT_mask_without_lashes_rotate(:,:,i) ;

end
```

Save Iris Segmentation Data

```
    % create the structure for saving
    seg6(number).original_image = iris_image;
    seg6(number).center_point = [m_pupil_center, n_pupil_center];
    seg6(number).pupil_radius = pupil_radius;
    seg6(number).iris_radius = iris_radius;
    seg6(number).mask_without_lashes = total_mask;
    seg6(number).mask_with_lashes = iris_mask;
    seg6(number).iris_with_lashes = iris_with_lashes;
    seg6(number).iris_without_lashes = isolated_iris;
```

```
    seg6(number).FT_original_image = FT_original_image(:,:,image_num);
    seg6(number).FT_mask_without_lashes = FT_mask_no_lashes(:,:,image_num);
    seg6(number).FT_mask_with_lashes = FT_mask_lashes(:,:,image_num);
    seg6(number).FT_iris_with_lashes = FT_iris_with_lashes(:,:,image_num);
    seg6(number).FT_mask_without_lashes = FT_iris_without_lashes(:,:,image_num);
    seg6(number).FT_iris_with_lashes_minus_mask = FT_iris_lashes(:,:,image_num);
    seg6(number).FT_iris_without_lashes_minus_mask = FT_iris_no_lashes(:,:,image_num);
    seg6(number).FT_iris_with_lashes_rotate_minus_mask = FT_iris_with_lashes_rotate_minus_mask;
    seg6(number).FT_iris_without_lashes_rotate_minus_mask =
FT_iris_without_lashes_rotate_minus_mask;
```

separates images into testing, training, and validation images

```
    test_num = [6, (13:7:700)];

    if ismember(number, test_num)
        seg(number).test = FT_iris_with_lashes_rotate_minus_mask(:,:,1);
    else
        seg(number).train = FT_iris_with_lashes_rotate_minus_mask;
        seg(number).validation = FT_iris_with_lashes;
    end
```

---

```
end
```

saves images of isolated iris for future use

```
filename=['C:\Users\Kevin\Documents\College\Graduate School\Thesis Iris Biometric\Finals Week
Combination\','isolated_iris.mat'];
%save(filename,'isolated');

toc
```

LOCATE PUPIL

```
% [m_pupil_center, n_pupil_center, radius, M_pupil_locations,
N_pupil_locations,iris_image_circled_pupil] = locate_pupil(iris_image,BLOCK)
% % INPUTS:
%     1.  iris_image = image of eye
% % OUTPUT:
%     1.  [M_pupil_locations,N_pupil_locations] = location of all pixels inside the circle
%     (list)
%     3.  iris_image_circled_pupil = iris_image with a white circle around
%     pupil
%     4. [m_pupil_center,n_pupil_center] = location of pupil center
%     5. radius = the radius of the pupil
% % FUNCTION:
%     1.  finds the pupil of an eye image
function [m_pupil_center, n_pupil_center, radius, M_pupil_locations,...
    N_pupil_locations,iris_image_circled_pupil] = locate_pupil(iris_image,BLOCK)
```

---

```
[M,N] = size(iris_image);
```

find intensity values in squares throughout the image

```
search_image = iris_image;
dark_index = 1;
    for j = BLOCK+1:BLOCK:N-BLOCK-1 %length
        for i = BLOCK+1:BLOCK:M-BLOCK-1 %height
            current_square = iris_image(i-BLOCK:i+BLOCK,j-BLOCK:j+BLOCK);
            squar(dark_index) = sum(sum(current_square));
            index_i(dark_index) = i;
            index_j(dark_index) = j;
            dark_index = dark_index+1;
        end
    end
```

locate the three darkest squares and the darkest pixel in each square

```
    first_location = find(squar == min(squar));
    first_location = first_location(1);
    m1 = index_i(first_location);
    n1 = index_j(first_location);
        squar(first_location) = max(squar);

    second_location = find(squar == min(squar));
    second_location = second_location(1);
    m2 = index_i(second_location);
    n2 = index_j(second_location);
        squar(second_location) = max(squar);

    third_location = find(squar == min(squar));
    third_location = third_location(1);
    m3 = index_i(third_location);
    n3 = index_j(third_location);
```

find a pixel in each dark square surrounded by other dark pixels

```
    [m1,n1] = ideal_location(m1,n1,iris_image,BLOCK);
    [m2,n2] = ideal_location(m2,n2,iris_image,BLOCK);
    [m3,n3] = ideal_location(m3,n3,iris_image,BLOCK);

% find the center of the pupil and the radius
    [radius1, m_cent1, n_cent1] = find_pupil(m1,n1, iris_image,BLOCK);
    [radius2, m_cent2, n_cent2] = find_pupil(m2,n2, iris_image,BLOCK);
    [radius3, m_cent3, n_cent3] = find_pupil(m3,n3, iris_image,BLOCK);

    radius = mean([radius1, radius2, radius3]);
    m_cent = round(mean([m_cent1, m_cent2, m_cent3]));
    n_cent = round(mean([n_cent1, n_cent2, n_cent3]));

% find circle around pupil
    [disk,pupil_m,pupil_n] = circle(M,N,m_cent,n_cent,radius);
    disk(m_cent,n_cent) = 255;

% OUTPUTS
    iris_image_circled_pupil = uint8(iris_image) + uint8(disk);
    M_pupil_locations = pupil_m;
    N_pupil_locations = pupil_n;
    radius = radius;
    m_pupil_center = m_cent;
    n_pupil_center = n_cent;
```

end

IDEAL LOCATION

```
% [m1,n1] = ideal_location(m1,n1,iris_image,BLOCK)
% % INPUTS:
%     1.  [m1,n1] = location of the first pixel of darkest square
%     3.  iris_image = original iris image
%     4.  BLOCK = size or block considered when finding darkest square
% % OUTPUT:
%     1. [m1,n1] = location of a pixel with dark neighbors (not on edge)
% % FUNCTION:
%     1.  finds a location that has dark spaces at its nearest 4 neighbors
function [out_m,out_n] = ideal_location(m1,n1,iris_image,BLOCK)

% shorten name
   iris = iris_image;

% locate the square in the iris image
   current_square = iris_image(m1-BLOCK:m1+BLOCK,n1-BLOCK:n1+BLOCK);

% cacluate the average pixel value in the dark square and give 10 percent
   avg = mean(mean(current_square));
   avg10 = avg +avg.*.1;

% find location with dark 4 neighbors
   for j = n1-BLOCK+1:n1+BLOCK-1
      for i = m1-BLOCK+1:m1+BLOCK-1
         if (iris(i-1,j)<=avg10 && iris(i+1,j)<=avg10 && ...
            iris(i,j-1)<=avg10 && iris(i,j+1)<=avg10 &&...
            iris(i-2,j)<=avg10 && iris(i+2,j)<=avg10 && ...
            iris(i,j-2)<=avg10 && iris(i,j+2)<=avg10)
            out_m = i;
            out_n = j;
            break
         else
         end
      end
   end
end
```

FIND PUPIL

```
% [radius1, m_cent1, n_cent1] = find_pupil(m,n, iris_image);
% % INPUTS:
%     1.  [m,n] location of a pixel with dark neighbors (not on edge of pupil)
%     2.  iris_image =  original image
% % OUTPUT:
%     1.  radius1 = radius of extimated pupil)
%     2.  [m_cent1,n_cent1] = center pixel of extimated pupil
% % FUNCTION:
%     1. find the center of the pupil and the radius given a point inside the
%        pupil
function [radius, m_cent, n_cent] = find_pupil(M,N, iris_image, BLOCK)
```

```
    % rename input variable as temp variable for editing
    image1 = iris_image;
    current_square = iris_image(M-BLOCK:M+BLOCK,N-BLOCK:N+BLOCK);
```

calculate the average pixel value in the dark square and give 10 percent

```
    avg = mean(mean(current_square));
    avg10 = avg +avg.*.1;

    [m_end,n_end] = size(iris_image);
    BLOCK = floor(BLOCK./2);
```

find 4 points on the pupil boundary

```
% search right for pupil boundary
for n = N:n_end
    array = iris_image(M, n-BLOCK:n+BLOCK);
    array = double(array);
    array_med = median(array);
    array_med = array_med(1);
        if array_med >= avg10
            right_local = n;
            break
        end
end

% search left for pupil boundary
for n = N:-1:1
    array = iris_image(M, n-BLOCK:n+BLOCK);
    array = double(array);
    size(array);
    array_med = median(array);
        if array_med >= avg10
            left_local = n;
            break
        end
end

% search up for pupil boundary
for m = M:-1:1
```

```
    array = iris_image(m-BLOCK:m+BLOCK,N);
    array = double(array);
    array_med = median(array);
        if array_med >= avg10
            top_local = m;
            break
        end
end

% search down for pupil boundary
for m = M:m_end
    array = iris_image(m-BLOCK:m+BLOCK,N);
    array = double(array);
    array_med = median(array);
        if array_med >= avg10
            bottom_local = m;
            break
        end
end
```

find center location

```
    m_cent = round((top_local + bottom_local)./2);
    n_cent = round((left_local + right_local)./2);
    image1(m_cent,n_cent) = 255;
```

find the average radius from center point to boundary points

```
    dist1 = pdist([m_cent,n_cent;top_local,N]);
    dist2 = pdist([m_cent,n_cent;bottom_local,N]);
    dist3 = pdist([m_cent,n_cent;M,left_local]);
    dist4 = pdist([m_cent,n_cent;M,right_local]);
    radius = mean([dist1,dist2,dist3,dist4]);
```

```
end
```

CIRCLE

```
% function [circle,m,n] = circle(M,N,center_m,center_n,radius)
% % INPUTS:
%     1.  M = height of image (pixels)
%     2.  N = width of image (pixels)
%     3.  center_m = height from origin to the center of  desired circle
%     4.  center_n = width from origin to the center of desired circle
%     5.  radius radius of desired circle (pixels)
% % OUTPUT:
%     1.  circle = image of size M X N with a value of 0 everywhere but the
%         desired circle (circle has values of 255)
%     2.  [m,n] = location of all pixels inside the circle
% % FUNCTION:
%     1.  draws a circle given the center and its radius

function [circ,m,n] = circle(M,N,center_m,center_n,radius)
   draw_circle = zeros(M+M,N+N);

   m=0;n =0;

   ro = sqrt(center_m^2 + center_n^2);
   R = radius;
   ango = 2*pi-atan(center_m/center_n);

   A = 1;
   C = ro^2 - R^2;

   for degree = 270:.001:360
      rad = degree *pi/180;
      B = -2*ro*cos(rad - ango);
      check = B^2 - 4*A*C;

      if check <0
         r_solution1 = NaN;
         r_solution2 = NaN;
      else
         r_solution1 = (-B + sqrt(check))/2;
         r_solution2 = (-B - sqrt(check))/2;

         m1 = -r_solution1*sin(rad);
         n1 = r_solution1*cos(rad);

         m2 = -r_solution2*sin(rad);
         n2 = r_solution2*cos(rad);

         draw_circle(ceil(m1),ceil(n1)) = 255;
         draw_circle(ceil(m2),ceil(n2)) = 255;
      end
   end

   pupil_index = 1;
for i = 1:N
   for j = 1:M
      dist = sqrt( (j-center_m).^2 + (i-center_n).^2);
```

```
        if dist <=radius;
            m(pupil_index) = j;
            n(pupil_index) = i;
            pupil_index = pupil_index+1;
        end

    end
end
circ(1:M,1:N) = draw_circle(1:M,1:N);

end
```

LOCATE IRIS

```
% function [iris_image_circled_iris, radius, m_iris, n_iris]
=locate_iris(image,edge_image,pupil_center_m, pupil_center_n, pupil_radius)
% % INPUTS:
%     1.  image = original eye image
%     2.  edge_image = eye image that only has edges =255, else = 0
%     3.  [pupil_center_m,pupil_center_n] = location of pupil center
%     4.  pupil_radius = radius of pupil
% % OUTPUT:
%     1.  iris_image_circled_iris = original eye image with white circle of
%     iris
%     2.  [m_iris,n_iris] = location of all pixels inside the iris (list)
%     3.  radius = radius of iris
% % FUNCTION:
%     1.  finds the iris of an eye image
function [iris_image_circled_iris, radius, m_iris, n_iris] =...
    locate_iris(image,edge_image,pupil_center_m, pupil_center_n, pupil_radius)
```

force inputs to become integers

```
pupil_center_m = round(pupil_center_m);
pupil_center_n = round(pupil_center_n);
pupil_radius = round(pupil_radius);

[M,N] = size(image);
possible_radius = pupil_radius+40:1:pupil_radius+80; %range determined from experimenting
```

delete edges that are not part of iris boundary

```
    edge_image(1:round(pupil_center_m-pupil_radius),1:N) = 0;
    edge_image(1:M,pupil_center_n-pupil_radius-10:pupil_center_n+pupil_radius+10) = 0;
```

Get rid of vertical edges

```
    vert_edge_kernel = [-1,0,1;-2,0,2;-3,0,3;-4,0,4;-3,0,3;-2,0,2;-1,0,1;];
    edge_image = conv2(double(edge_image), double(vert_edge_kernel),'same');
    edge_image = edge_image > 2;
    edge_image = bwareaopen(edge_image, 50);
```

search for circles using a circular hough transform

```
    h = circle_hough(edge_image, possible_radius, 'same', 'normalise');
    peaks = circle_houghpeaks(h, possible_radius, 'nhoodxy', 15, 'nhoodr', 21, 'npeaks', 1);
```

find circle around iris

```
    [disk,m_iris,n_iris] = circle(M,N,pupil_center_m,pupil_center_n,round(peaks(3)));
    disk(pupil_center_m,pupil_center_n) = 255;
```

OUTPUTS

```
    iris_image_circled_iris = uint8(image) + uint8(disk);
    m_iris = m_iris;
    n_iris = n_iris;
    radius = peaks(3);
```

end

CIRCLE HOUGH

```
function [h, margin] = circle_hough(b, rrange, varargin)
%CIRCLE_HOUGH Hough transform for circles
%   [H, MARGIN] = CIRCLE_HOUGH(B, RADII) takes a binary 2-D image B and a
%   vector RADII giving the radii of circles to detect. It returns the 3-D
%   accumulator array H, and an integer MARGIN such that H(I,J,K) contains
%   the number of votes for the circle centred at B(I-MARGIN, J-MARGIN),
%   with radius RADII(K). Circles which pass through B but whose centres
%   are outside B receive votes.
%
%   [H, MARGIN] = CIRCLE_HOUGH(B, RADII, opt1, ...) allows options to be
%   set. Each option is a string, which if included has the following
%   effect:
%
%   'same' returns only the part of H corresponding to centre positions
%   within the image. In this case H(:,:,k) has the same dimensions as B,
%   and MARGIN is 0. This option should not be used if circles whose
%   centres are outside the image are to be detected.
%
%   'normalise' multiplies each slice of H, H(:,:,K), by 1/RADII(K). This
%   may be useful because larger circles get more votes, roughly in
%   proportion to their radius.
%
%   The spatial resolution of the accumulator is the same as the spatial
%   resolution of the original image. Smoothing the accumulator array
%   allows the effective resolution to be controlled, and this is probably
%   essential for sensitivity to circles of arbitrary radius if the spacing
%   between radii is greater than 1. If time or memory requirements are a
%   problem, a generalisation of this function to allow larger bins to be
%   used from the start would be worthwhile.
%
%   Each feature in B is allowed 1 vote for each circle. This function
%   could easily be generalised to allow weighted features.
%
%   See also CIRCLEPOINTS, CIRCLE_HOUGHPEAKS, CIRCLE_HOUGHDEMO

% Copyright David Young 2008, 2010

% argument checking
opts = {'same' 'normalise'};
error(nargchk(2, 2+length(opts), nargin, 'struct'));
validateattributes(rrange, {'double'}, {'real' 'positive' 'vector'});
if ~all(ismember(varargin, opts))
   error('Unrecognised option');
end

% get indices of non-zero features of b
[featR, featC] = find(b);

% set up accumulator array - with a margin to avoid need for bounds checking
[nr, nc] = size(b);
nradii = length(rrange);
margin = ceil(max(rrange));
nrh = nr + 2*margin;        % increase size of accumulator
```

```
nch = nc + 2*margin;
h = zeros(nrh*nch*nradii, 1, 'uint32');  % 1-D for now, uint32 a touch faster

% get templates for circles at all radii - these specify accumulator
% elements to increment relative to a given feature
tempR = []; tempC = []; tempRad = [];
for i = 1:nradii
   [tR, tC] = circlepoints(rrange(i));
   tempR = [tempR tR]; %#ok<*AGROW>
   tempC = [tempC tC];
   tempRad = [tempRad repmat(i, 1, length(tR))];
end

% Convert offsets into linear indices into h - this is similar to sub2ind.
% Take care to avoid negative elements in either of these so can use
% uint32, which speeds up processing by a factor of more than 3 (in version
% 7.5.0.342)!
tempInd = uint32( tempR+margin + nrh*(tempC+margin) + nrh*nch*(tempRad-1) );
featInd = uint32( featR' + nrh*(featC-1)' );

% Loop over features
for f = featInd
   % shift template to be centred on this feature
   incI = tempInd + f;
   % and update the accumulator
   h(incI) = h(incI) + 1;
end

% Reshape h, convert to double, and apply options
h = reshape(double(h), nrh, nch, nradii);

if ismember('same', varargin)
   h = h(1+margin:end-margin, 1+margin:end-margin, :);
   margin = 0;
end

if ismember('normalise', varargin)
   h = bsxfun(@rdivide, h, reshape(rrange, 1, 1, length(rrange)));
end

end
```

CIRCLE POINTS

```
function [x, y] = circlepoints(r)
%CIRCLEPOINTS  Returns integer points close to a circle
%   [X, Y] = CIRCLEPOINTS(R) returns coordinates of integer points close to
%   a circle of radius R, such that none is repeated and there are no gaps
%   in the circle (under 8-connectivity).

%   Copyright David Young 2010

% Get number of rows needed to cover 1/8 of the circle
l = round(r/sqrt(2));
if round(sqrt(r.^2 - l.^2)) < l   % if crosses diagonal
   l = l-1;
end
% generate coords for 1/8 of the circle, a dot on each row
x0 = 0:l;
y0 = round(sqrt(r.^2 - x0.^2));
% Check for overlap
if y0(end) == l
   l2 = l;
else
   l2 = l+1;
end
% assemble first quadrant
x = [x0 y0(l2:-1:2)];
y = [y0 x0(l2:-1:2)];
% add next quadrant
x0 = [x y];
y0 = [y -x];
% assemble full circle
x = [x0 -x0];
y = [y0 -y0];
end
```

CIRCLE HOUGH PEAKS

```
function peaks = circle_houghpeaks(h, radii, varargin)
%CIRCLE_HOUGHPEAKS finds peaks in the output of CIRCLE_HOUGH
%   PEAKS = CIRCLE_HOUGHPEAKS(H, RADII, MARGIN, OPTIONS) locates the
%   positions of peaks in the output of CIRCLE_HOUGH. The result PEAKS is a
%   3 x N array, where each column gives the position and radius of a
%   possible circle in the original array. The first row of PEAKS has the
%   x-coordinates, the second row has the y-coordinates, and the third row
%   has the radii.
%
%   H is the 3D accumulator array returned by CIRCLE_HOUGH.
%
%   RADII is the array of radii which was passed as an argument to
%   CIRCLE_HOUGH.
%
%   MARGIN is optional, and may be omitted if the 'same' option was used
%   with CIRCLE_HOUGH. Otherwise, it should be the second result returned
%   by CIRCLE_HOUGH.
%
%   OPTIONS is a comma-separated list of parameter/value pairs, with the
%   following effects:
%
%   'Smoothxy' causes each x-y layer of H to be smoothed before peak
%   detection using a 2D Gaussian kernel whose "sigma" parameter is given
%   by the value of this argument.
%
%   'Smoothr' causes each radius column of H to be smoothed before peak
%   detection using a 1D Gaussian kernel whose "sigma" parameter is given
%   by the value of this argument.
%
%      Note: Smoothing may be useful to locate peaks in noisy accumulator
%      arrays. However, it may also cause the performance to deteriorate
%      if H contains sharp peaks. It is most likely to be useful if
%      neighbourhood suppression (see below) is not used.
%
%      Both smoothing operations use reflecting boundary conditions to
%      compute values close to the boundaries.
%
%   'Threshold' sets the minimum number of votes (after any smoothing)
%   needed for a peak to be counted. The default is 0.5 * the maximum value
%   in H.
%
%   'Npeaks' sets the maximum number of peaks to be found. The highest
%   NPEAKS peaks are returned, unless the threshold causes fewer than
%   NPEAKS peaks to be available.
%
%   'Nhoodxy' must be followed by an odd integer, which sets a minimum
%   spatial separation between peaks. See below for a more precise
%   statement. The default is 1.
%
%   'Nhoodr' must be followed by an odd integer, which sets a minimum
%   separation in radius between peaks. See below for a more precise
%   statement. The default is 1.
%
```

```
%       When a peak has been found, no other peak with a position within an
%       NHOODXY x NHOODXY x NHOODR box centred on the first peak will be
%       detected. Peaks are found sequentially; for example, after the
%       highest peak has been found, the second will be found at the
%       largest value in H excepting the exclusion box found the first
%       peak. This is similar to the mechanism provided by the Toolbox
%       function HOUGHPEAKS.
%
%       If both the 'Nhoodxy' and 'Nhoodr' options are omitted, the effect
%       is not quite the same as setting each of them to 1. Instead of a
%       sequential algorithm with repeated passes over H, the Toolbox
%       function IMREGIONALMAX is used. This may produce slightly different
%       results, since an above-threshold point adjacent to a peak will
%       appear as an independent peak using the sequential suppression
%       algorithm, but will not be a local maximum.
%
%   See also CIRCLE_HOUGH, CIRCLE_HOUGHDEMO

% check arguments
params = checkargs(h, radii, varargin{:});

% smooth the accumulator - xy
if params.smoothxy > 0
    [m, hsize] = gaussmask1d(params.smoothxy);
    % smooth each dimension separately, with reflection
    h = cat(1, h(hsize:-1:1,:,:), h, h(end:-1:end-hsize+1,:,:));
    h = convn(h, reshape(m, length(m), 1, 1), 'valid');

    h = cat(2, h(:,hsize:-1:1,:), h, h(:,end:-1:end-hsize+1,:));
    h = convn(h, reshape(m, 1, length(m), 1), 'valid');
end

% smooth the accumulator - r
if params.smoothr > 0
    [m, hsize] = gaussmask1d(params.smoothr);
    h = cat(3, h(:,:,hsize:-1:1), h, h(:,:,end:-1:end-hsize+1));
    h = convn(h, reshape(m, 1, 1, length(m)), 'valid');
end

% set threshold
if isempty(params.threshold)
    params.threshold = 0.5 * max(h(:));
end

if isempty(params.nhoodxy) && isempty(params.nhoodxy)
    % First approach to peak finding: local maxima

    % find the maxima
    maxarr = imregionalmax(h);

    maxarr = maxarr & h >= params.threshold;

    % get array indices
    peakind = find(maxarr);
    [y, x, rind] = ind2sub(size(h), peakind);
    peaks = [x'; y'; radii(rind)];
```

```
    % get strongest peaks
    if ~isempty(params.npeaks) && params.npeaks < size(peaks,2)
        [~, ind] = sort(h(peakind), 'descend');
        ind = ind(1:params.npeaks);
        peaks = peaks(:, ind);
    end

else
    % Second approach: iterative global max with suppression
    if isempty(params.nhoodxy)
        params.nhoodxy = 1;
    elseif isempty(params.nhoodr)
        params.nhoodr = 1;
    end
    nhood2 = ([params.nhoodxy params.nhoodxy params.nhoodr]-1) / 2;

    if isempty(params.npeaks)
        maxpks = 0;
        peaks = zeros(3, round(numel(h)/100));  % preallocate
    else
        maxpks = params.npeaks;
        peaks = zeros(3, maxpks);  % preallocate
    end

    np = 0;
    while true
        [r, c, k, v] = max3(h);
        % stop if peak height below threshold
        if v < params.threshold || v == 0
            break;
        end
        np = np + 1;
        peaks(:, np) = [c; r; radii(k)];
        % stop if done enough peaks
        if np == maxpks
            break;
        end
        % suppress this peak
        r0 = max([1 1 1], [r c k]-nhood2);
        r1 = min(size(h), [r c k]+nhood2);
        h(r0(1):r1(1), r0(2):r1(2), r0(3):r1(3)) = 0;
    end
    peaks(:, np+1:end) = [];   % trim
end

% adjust for margin
if params.margin > 0
    peaks([1 2], :) = peaks([1 2], :) - params.margin;
end
end

function params = checkargs(h, radii, varargin)
% Argument checking
ip = inputParser;
```

```
% required
htest = @(h) validateattributes(h, {'double'}, {'real' 'nonnegative' 'nonsparse'});
ip.addRequired('h', htest);
rtest = @(radii) validateattributes(radii, {'double'}, {'real' 'positive' 'vector'});
ip.addRequired('radii', rtest);

% optional
mtest = @(n) validateattributes(n, {'double'}, {'real' 'nonnegative' 'integer' 'scalar'});
ip.addOptional('margin', 0, mtest);

% parameter/value pairs
stest = @(s) validateattributes(s, {'double'}, {'real' 'nonnegative' 'scalar'});
ip.addParamValue('smoothxy', 0, stest);
ip.addParamValue('smoothr', 0, stest);
ip.addParamValue('threshold', [], stest);
nptest = @(n) validateattributes(n, {'double'}, {'real' 'positive' 'integer' 'scalar'});
ip.addParamValue('npeaks', [], nptest);
nhtest = @(n) validateattributes(n, {'double'}, {'odd' 'positive' 'scalar'});
ip.addParamValue('nhoodxy', [], nhtest);
ip.addParamValue('nhoodr', [], nhtest);
ip.parse(h, radii, varargin{:});
params = ip.Results;
end

function [m, hsize] = gaussmask1d(sigma)
% truncated 1D Gaussian mask
hsize = ceil(2.5*sigma);  % reasonable truncation
x = (-hsize:hsize) / (sqrt(2) * sigma);
m = exp(-x.^2);
m = m / sum(m);  % normalise
end

function [r, c, k, v] = max3(h)
% location and value of global maximum of a 3D array
[vr, r] = max(h);
[vc, c] = max(vr);
[v, k] = max(vc);
c = c(1, 1, k);
r = r(1, c, k);
end
```

## CREATE FOURIER COEFFICIENTS

```
clear all; close all; clc;
filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris
Biometric_for_writeup\Finals Week Combination\','isolated_iris.mat'];
load(filename);
load eye_assignments % groups the eyes together (each prson has 7 eyes)

num_coefficients  = 121;
ROTATION = 0;
noise = 'gaussian'; %'gaussian', 'saltpepper', or 'none'
snr = 1;
```

Finds the location of coefficients that are to be kept

```
length_of_side = round(sqrt(num_coefficients));
N = floor(length_of_side./2);
M = N;
[X,Y] = deal(280,320);
middle = zeros(X,Y);
middle(round(X/2-N:X/2+N),round(Y/2-M:Y/2+M)) = 1;
coef_location = find(middle ==1);
```

FT: take FT of all images

```
% check to see if noise is being applied to the images
   if strcmp('none',noise)
      % do  nothing
   else
      load Gaussian.mat
      load SaltPepper.mat
   end

   [location,eye] = deal(0,1);
   actual_snr = zeros(1,700);
   actual_sp_mask_snr = zeros(1,700);
   FT_coef = zeros(num_coefficients,700);

   for num = 1:700
      im = imrotate(isolated(num).lashes,ROTATION,'bilinear','crop');

      % add noise to the image
      if strcmp('gaussian',noise)
         [g_minimum, g_min_loc] = min(abs(gaussian_noisedata(:,2)-snr));
         g_intensity = gaussian_noisedata(g_min_loc,1);
         g_noisy = imnoise(im,'Gaussian',0,g_intensity);
         image = g_noisy;
      elseif strcmp('saltpepper',noise)
         [sp_minimum, sp_min_loc] = min(abs(s_p_noisedata(:,2)-snr));
         sp_intensity = s_p_noisedata(sp_min_loc,1);
         sp_noisy = imnoise(im,'Salt & Pepper',sp_intensity);
         image = sp_noisy;
      else  % add no noise to the image
         image = im;
```

```
        end

        mask = uint8(isolated(num).mask);
        noisy_image_re_mask = mask .* image;

        % find the SNR of new, noisy image
        actual_snr(num) = SNR(im,image);
        actual_sp_mask_snr(num) = SNR(im,noisy_image_re_mask);

        location = location+1;

        rotated_image = imrotate(image,ROTATION,'bilinear','crop');
        FT_coef(:,num) = real(get_coef(isolated(num).lashes,isolated(num).mask,coef_location));

    end
```

Normalize the FT coefficients

```
    maximum = 10267;
    minimum = -8674;
    FT_values = [(FT_coef-minimum)./(maximum-minimum)];

    FT_weights = [eye_assignments;FT_values];
```

Save coefficients for future use

```
if ROTATION == 0
    filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','FT_coefficients.mat'];
    save(filename,'FT_weights')
else
    filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','FT_coefficients_rotated.mat'];
    save(filename,'FT_weights')
end

disp(sprintf('DONE with FT rotation= %.0f',ROTATION))
```

SNR

```
% function snr_power = SNR(image, noisy_image)
% % INPUTS:
%     1.  image = image to compare added noise with
%     2.  noisy_image = image to find SNR
% % OUTPUT:
%     1.  snr_power = Signal to Noise ratio (dB)
% % FUNCTION:
%     1.  finds the signal to noise ratio of a noisy image

function snr_power = SNR(image, noisy_image)
% SNR (Signal to noise ratio)
   noise = abs(image - noisy_image);

   signalAmp = image(:);
   noiseAmp = noise(:);

   signalPower = sum(signalAmp.^2)/numel(image);
   noisePower = sum(noiseAmp.^2)/numel(noise);

   snr_power = 10*log10(signalPower/noisePower);
end
```

GET COEFFICIENTS

```
% function [ coefficients ] = get_coef(image,mask,coef_locations)
% % INPUTS:
%     1.  image = iris image
%     2.  mask = mask used to get iris image
%     3.  coef_locations = location of FT coefficietnbts that are to be
%     saved
% % OUTPUT:
%     1.  coefficients = FT coefficients of specified locations
% % FUNCTION:
%     1.  takes in an eye image that has the mask applied and the mask that was
%       applied.  Takes the Fourier Transform of the 2 and returns coefficients
%       of the transform

function [ coefficients ] = get_coef(image,mask,coef_locations)

% take FT of image and mask
   FT_image = fft2(image);
   FT_mask = fft2(mask);

% subtrack the FT due to mask from FT of image
   FT_diff = FT_image - FT_mask;

% return the coeffieients in teh locations specified by input
   coefficients = FT_diff(coef_locations);
end
```

## CREATE EIGEN EYE COEFFICIENTS

```
clear all; close all; clc;
tic
```

load in eye images to have their Eigen coefficients found

```
filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','isolated_iris.mat'];
load(filename);
ROTATION = 0;
```

Declare the basis images

```
SAMPLES = 600:699;
noise = 'saltpepper'; %gaussian saltpepper or none
snr = 20;
```

read in the basis images and convert to columns and make average eye

```
    samples = [];
    for i = SAMPLES
        sample_image = isolated(i).lashes;
        [M,N] = size(sample_image);
        linear_sample = reshape(sample_image,M*N,1);
        samples = [samples ,linear_sample];
    end
```

find the mean and std deviation of the samples (used for normalization)

```
    [sample_sum, sample_std] = deal(0);
    [l,w] = size(samples);
    sample_mean = sum(sum(samples))/(l*w);

    for i = 1:length(SAMPLES)
        current_sample = double(samples(:,i));
        sample_std = sum((current_sample - sample_mean).^2) + sample_std;
    end
    sample_std = sqrt((sample_std./(length(SAMPLES)*M*N)));
```

Normalize samples to remove possible differences in lighting

```
    for i = 1:length(SAMPLES)
        current_mean = mean(double(samples(:,i)));
        current_std = std(double(samples(:,i)));
        samples(:,i) = double(samples(:,i) - current_mean)/current_std * sample_std + sample_mean;
    end
    samples = double(samples);
```

calculate the mean eye

```
avg_eye = mean(samples,2);
mean_eye = uint8(reshape(avg_eye,M,N));
```

Calculate the Eigen vectors and values

```
C = samples'*samples;
[init_vectors, init_values] = eig(C);
```

eliminate vectors who's values are essentially 0

```
[e_vectors,e_values] = deal([]);
for i = 1:length(SAMPLES)
    if init_values(i,i) > 1e-4
        e_vectors = [e_vectors init_vectors(:,i)];
        e_values = [e_values init_values(i,i)];
    end
end
```

put Eigen values in descending order and rearrange vectors the same way

```
[eigen_values, ORDER] = sort(e_values,'descend');
for i = 1:size(ORDER,2)
    eigen_vectors(:,i) = e_vectors(:,ORDER(i));
end
```

Normalize since actual covariance matrix was not used

```
for i = 1:size(eigen_vectors,2)
    eigen_vectors(:,i) = eigen_vectors(:,i)./(sum(eigen_vectors(:,i).^2).^2);
    eigenvectors(:,i) = samples*eigen_vectors(:,i);
end
```

Apply possible noise to image

```
% check to see if noise is being applied to the images
    if strcmp('none',noise)
        % do  nothing
    else
        load Gaussian.mat %gaussian_noisedata
        load SaltPepper.mat %s_p_noise_data
        load NEWgaussian.mat %actual_snr
    end
```

find the coefficients of input eyes

```
for number = 1:700
```

```
% tag eye with group (1 to 100)
    if number >=7
        tag(number) = floor(number./7)+1;
    else
```

```
    tag(number) = 1;
end

% take in an image
im = imrotate(isolated(number).lashes,ROTATION,'bilinear','crop');

% add noise to the image
if strcmp('gaussian',noise)
    [g_minimum, g_min_loc] = min(abs(actual_snr(:,2)-snr));
    g_intensity = actual_snr(g_min_loc,1);
    H = fspecial('disk',g_intensity);
    blur = imfilter(im,H,'replicate');
    image = blur;
elseif strcmp('saltpepper',noise)
    [sp_minimum, sp_min_loc] = min(abs(s_p_noisedata(:,2)-snr));
    sp_intensity = s_p_noisedata(sp_min_loc,1);
    sp_noisy = imnoise(im,'Salt & Pepper',sp_intensity);
    image = sp_noisy;

else % add no noise to the image
    image = im;


end

S_N_R(number) = SNR(im,image);

input_eye = double(reshape(imresize(image,[M N]),M*N,1));
input_mean = mean(input_eye);
input_std = std(input_eye);
```

normalize the input image

```
input_eye = (input_eye-input_mean)/input_std * sample_std + sample_mean;
eye_diff = input_eye - avg_eye;
```

Get Weights

```
weights = double(zeros(length(SAMPLES),1));
for i = 1:length(weights)
  eigen_coef(i,number) = dot(eigenvectors(:,i)',eye_diff');
end
```

---

```
end
```

attach the tag to the weights as the first coefficient

```
maximum = 2.1524e+09;
minimum = -3.8845e+08;

      absolute_max = 2.1524e9;
eigen_weights = [tag;(eigen_coef-minimum)./(maximum-minimum)];
```

Save coefficients

```
    if ROTATION ==0
        filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\EigenCoeff\','eigen_weights.mat'];
        save(filename,'eigen_weights');
    else
        filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\EigenCoeff\','eigen_weights_rotated.mat'];
        save(filename,'eigen_weights');
    end


disp(sprintf('DONE with Eigen rotation = %.0f and SNR = %.3f',ROTATION,mean(S_N_R)))
```

## CREATE DISCRETE COSINE COEFFICIENTS

```
clear all; close all; clc;
tic
```

Apply possible noise to images

```
ROTATION = 0;
noise = 'none'; %gaussian saltpepper or none
snr = 20;

filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','isolated_iris.mat'];
load(filename);
load eye_assignments.mat;

% load appropriate imges (based on noise)
   if strcmp('none',noise)
      % do  nothing
   else
      load Gaussian.mat
      load SaltPepper.mat
      load NEWgaussian.mat
   end
```

location of DCT coefficients to be used

```
M = 1:10;
N = 1:10;
```

apply noise to input images

```
   [location,eye] = deal(0,1);
   for num = 1:700
      im = imrotate(isolated(num).lashes,ROTATION,'bilinear','crop');
   if strcmp('gaussian',noise)
      [g_minimum, g_min_loc] = min(abs(actual_snr(:,2)-snr));
      g_intensity = actual_snr(g_min_loc,1);
      H = fspecial('disk',g_intensity);
      blur = imfilter(im,H,'replicate');
      image = blur;
   elseif strcmp('saltpepper',noise)
      [sp_minimum, sp_min_loc] = min(abs(s_p_noisedata(:,2)-snr));
      sp_intensity = s_p_noisedata(sp_min_loc,1);
      sp_noisy = imnoise(im,'Salt & Pepper',sp_intensity);
      image = sp_noisy;
   else
      % add no noise to the image
      image = im;
   end

   S_N_R(num) = SNR(im,image);
   this = dct2(im);
```

```
now = zeros(280,320);
now(M,N) = this(M,N);
what = idct2(now)./255;
save_these_weights = now(M,N);

DCT_values(:,num) = save_these_weights(:);

end

maximum = 1.4977e+03;
average = 2.1248;
abs_max = 1.9718e3;
minimum = -1.9718e+03;

DCT_values = [(DCT_values-average)./abs_max]; % used for tansig activation
DCT_weights = [eye_assignments;DCT_values];
```

## CREATE WAVELET COEFFICIENTS

```
clear all; close all;
```

Assign possible noise to input images

```
ROTATION = 0;
noise = 'none'; %gaussian saltpepper or none
snr = 5;
```

Load the iris images

```
filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','isolated_iris.mat'];
load(filename);
load eye_assignments.mat;

   if strcmp('none',noise)
      % do  nothing
   else
      load Gaussian.mat
      load SaltPepper.mat
      load NEWgaussian.mat
   end
```

Take WVT of each image

```
WVTcoef = zeros(90,700);
for i = 1:700
   im = isolated(i).lashes;

% add noise to the image via look up table
   if strcmp('gaussian',noise)
      [g_minimum, g_min_loc] = min(abs(actual_snr(:,2)-snr));
      g_intensity = actual_snr(g_min_loc,1);
      H = fspecial('disk',g_intensity);
      blur = imfilter(im,H,'replicate');
      image = blur;

   elseif strcmp('saltpepper',noise)
      [sp_minimum, sp_min_loc] = min(abs(s_p_noisedata(:,2)-snr));
      sp_intensity = s_p_noisedata(sp_min_loc,1);
      sp_noisy = imnoise(im,'Salt & Pepper',sp_intensity);
      image = sp_noisy;
   else % add no noise to the image
      image = im;
   end

% apply rotation
   rotated_image = imrotate(image,ROTATION,'bilinear','crop');
   S_N_R(i) = SNR(im,rotated_image);

% Perform decomposition at level 5
```

```
    [c,s] = wavedec2(rotated_image,5,'db1');

% Extract approximation coefficients at level 2.
    WVT = appcoef2(c,s,'db5',5);
    WVT = WVT(:);
    WVTcoef(:,i) = WVT;

end
```

Normalize the coefficients

```
maximum_value = 7.8389e3;
WVTcoef = WVTcoef./maximum_value;


    WVT_weights = [eye_assignments;WVTcoef];

    if (ROTATION == 0)
        filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','WVT_coefficients.mat'];
        save(filename,'WVT_weights');
    else
        filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','WVT_coefficients_rotated.mat'];
        save(filename,'WVT_weights');
    end
toc
W_V_T = ROTATION;
signal_to_noise = mean(S_N_R)
```

## TRAIN NEURAL NETWORK

```
clear all; close all; clc;
```

Assign neural network parameters before training

```
LAYERS = [15 21 11];
ITERATIONS = 100000000; % max iterations
ACCEPTABLE_ERROR = .01;
BIAS = 1;
ALPHA = .8;
ETA = .5;
MOMENTUM = .4;
```

create PHI functions

```
PHI = @(v) 1./(1+exp(-ALPHA.*v));
PHIPRIME = @(v) ALPHA.*v.*(1-v);
```

Normalize Inputs and Create Targets Matrixes

```
% load eigen inputs
      filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\EigenCoeff\','eigen_weights.mat'];
      load(filename);

      [ validation_input_eigen, train_input_eigen, test_input_eigen,...
      validation_target_eigen, train_target_eigen, test_target_eigen,e_weights,e_targets,
normalization_factors ]...
      = set_eigen_inputs( eigen_weights );

% load FT coefficients
   % load FT inputs
      filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','FT_coefficients.mat'];
      load(filename);

      [ validation_input_FT, train_input_FT, test_input_FT,...
      validation_target_FT, train_target_FT, test_target_FT,FT_weights2,FT_targets,
normalization_factors ]...
      = set_eigen_inputs( FT_weights );

% load DCT coefficients
      filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','DCT_coefficients_rotated.mat'];
      load(filename);

      [ validation_input_DCT, train_input_DCT, test_input_DCT,...
      validation_target_DCT, train_target_DCT, test_target_DCT,DCT_weights2,DCT_targets,
normalization_factors ]...
      = set_eigen_inputs( DCT_weights );

% load Wavelet Coefficients
```

```
    filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals
Week Combination\','WVT_coefficients.mat'];
    load(filename)

    [ validation_input_WVT, train_input_WVT, test_input_WVT,...
    validation_target_WVT, train_target_WVT, test_target_WVT,WVT_weights2,WVT_targets,
normalization_factorsWVT ]...
    = set_eigen_inputs( WVT_weights );
```

Eye inputs

```
    num_coef = 121;
    num_outputs = 3;
    num_eyes = 7;
    total = num_coef*LAYERS(1);
    for i = 2:length(LAYERS)
        total = total +LAYERS(i-1)*LAYERS(i);
    end
    connections = total +sum(LAYERS);
```

Use only one combination of inputs

```
pick_type = 'W';
    if(pick_type == 'F')
        inputs = train_input_FT(1:end,1:num_eyes);
    elseif(pick_type == 'E')
        inputs = train_input_eigen(1:50,1:num_eyes);
    elseif(pick_type == 'D')
        inputs = train_input_DCT(1:num_coef,1:num_eyes);
    elseif(pick_type == 'W')
        inputs = train_input_WVT(:,1:num_eyes);
    elseif(pick_type == 'FE')
         inputs = [train_input_eigen(1:num_coef,1:num_eyes);train_input_FT(1:num_coef,1:num_eyes)];
    elseif(pick_type == 'FW')
        inputs = [train_input_FT(1:121,1:num_eyes);train_input_WVT(1:90,1:num_eyes)];
    elseif(pick_type == 'DE')
        inputs = [train_input_DCT(1:num_coef,1:num_eyes);train_input_eigen(1:50,1:num_eyes)];
    elseif(pick_type == 'WD')
        inputs = [train_input_DCT(:,1:num_eyes);train_input_WVT(:,1:num_eyes)];
    else
        return;
    end

    targets = train_target_FT(:,1:num_eyes);
    targets = targets((7-num_outputs+1):7,:);
    validation_in = inputs;
    validation_tar = targets;
```

Organize inputs

```
    [num_inputs,length_inputs] = size(inputs);
    [num_outputs,~] = size(targets);
    layers = [num_inputs,LAYERS,num_outputs];
```

Initialize weights for neural network (random)

```
[net_weights, net_dw, net_lastdw] = deal(zeros(max(layers)*max(layers),1));
for i = 2:length(layers)
   net_weights(1:(layers(i)*(layers(i-1)+1)),i-1) = rand(layers(i)*(layers(i-1)+1),1).*2-1;
   net_dw(1:(layers(i)*(layers(i-1)+1)),i-1) = zeros(layers(i)*(layers(i-1)+1),1);
   net_lastdw(1:(layers(i)*(layers(i-1)+1)),i-1) = rand(layers(i)*(layers(i-1)+1),1).*2-1;
end
net_lastdw = net_weights - net_lastdw;
mse= zeros(ITERATIONS,1);
```

Run Neural Network until it reaches max iterations or min error

```
count = 0;error = inf; location = 0; num_wrong=inf;
tic
while count < ITERATIONS && error > ACCEPTABLE_ERROR

% increase the momentum parameter after certain MSE
   if (num_wrong <1 && MOMENTUM<.7) || error< 1.1
      MOMENTUM = 0.9;
   end

% counters
   count = count+1;
   location = location+1;
   if location > length_inputs
      location = 1;
   end

% FORWARD PROPAGATION
   net_x = zeros(max(layers)+1,length(layers));
   net_x(1:length(inputs(:,location))+1,1) = [BIAS;inputs(:,location)];
   [net_v, net_e] = deal(zeros(max(layers),length(layers)));
   net_v(1:length(inputs(:,location))) = inputs(:,location);

   len = length(inputs(:,location))+1;
   for LAYER = 2:length(layers)
      num_w = layers(LAYER-1)+1;
         for NEURON = 1:layers(LAYER)
            net_v(NEURON,LAYER) = net_x(1:len,LAYER-1)'*...
               net_weights(NEURON*num_w-(num_w-1):NEURON*num_w,LAYER-1);
         end
      len = layers(LAYER)+1;
      net_x(1:len,LAYER)= [BIAS; PHI(net_v(1:len-1,LAYER))];
   end
%     fprintf('forward propagation time = %.4f ms\n',toc*1000)

% BACK PROPAGATION
   for LAYER = length(layers):-1:2
      if LAYER == length(layers) %disp('OUTPUT LAYER') % output neuron
         for NEURON = 1:layers(LAYER)
            e = targets(:,location) - net_x(2:len,LAYER);
            num_e = layers(LAYER);
            net_e(1:layers(LAYER),LAYER) = PHIPRIME(net_x(2:len,LAYER)).*e;
            net_dw(NEURON*num_w-(num_w-1):num_w*NEURON,LAYER-1) =...
```

```
                ETA.*net_e(NEURON,LAYER).*net_x(1:layers(LAYER-1)+1,LAYER-1)...
                +MOMENTUM.* net_lastdw(NEURON*num_w-(num_w-1):num_w*NEURON,LAYER-
1);
              net_lastdw(NEURON*num_w-(num_w-1):num_w*NEURON,LAYER-1) =
net_dw(NEURON*num_w-(num_w-1):num_w*NEURON,LAYER-1);
           end
        else % disp('HIDDEN LAYER')% hidden neuron
           num_neurons = layers(LAYER+1);
           jump = layers(LAYER)+1;
           last = num_neurons*jump;

           for NEURON = 1:layers(LAYER)
              m = net_weights(NEURON:jump:last);
              em = net_e(1:layers(LAYER+1),LAYER+1);
              g = m*em;
              net_e(NEURON,LAYER) = g*PHIPRIME(net_x(1+NEURON,LAYER));
              num_w = layers(LAYER-1)+1;
              net_dw(NEURON*num_w-(num_w-1):num_w*NEURON,LAYER-1)=...
                 ETA.*net_e(NEURON,LAYER).*net_x(1:layers(LAYER-1)+1,LAYER-1)...
                 + MOMENTUM .* net_lastdw(NEURON*num_w-(num_w-1):num_w*NEURON,LAYER-
1);
              net_lastdw(NEURON*num_w-(num_w-1):num_w*NEURON,LAYER-
1)=net_dw(NEURON*num_w-(num_w-1):num_w*NEURON,LAYER-1);
           end
        end
     end


% ADJUST WEIGHTS
     net_weights = net_weights + net_dw;  % weights + weight change
     [~,cases] = size(validation_in);
     [num_tar,~] = size(validation_tar);
     net_output = zeros(num_tar,cases);

% Calculate the MSE
     for test = 1:cases
        val_x = zeros(max(layers)+1,length(layers));
        val_x(1:length(validation_in(:,test))+1,1)= [BIAS; validation_in(:,test)];
        [val_v, val_e] = deal(zeros(max(layers),length(layers)));
        val_v(1:length(validation_in(:,test)))=validation_in(:,test);

        len = length(validation_in(:,test))+1;
        for LAYER = 2:length(layers)
           num_w = layers(LAYER-1)+1;
           for NEURON = 1:layers(LAYER);
              val_v(NEURON,LAYER) = val_x(1:len,LAYER-1)'*...
                 net_weights(NEURON*num_w-(num_w-1):NEURON*num_w,LAYER-1);
           end
           len = layers(LAYER)+1;
           val_x(1:len,LAYER)=[BIAS; PHI(val_v(1:len-1,LAYER))];
        end
        net_output(:,test) = val_x(2:1+num_tar,end);
     end

     net_output_round = round(net_output);
```

```
    d = validation_tar(:);
    y = net_output(:);
    error = mean(sum((d-y).^2));
    mse(count,1) = error;

    num_wrong = sum(sum(abs(abs(net_output_round)-abs(validation_tar))));

    if num_wrong == 0
        fprintf('ITERATION: %.0d\n',count)
        saved_weights = net_weights;
    end
end


toc
if count == ITERATIONS && num_wrong~=0
    color = 'r';
else
    color = 'b';
end

figure(2)
    plot(mse(1:count),color)
        xlabel('Iterations')
        ylabel('Percent Error')
        title(sprintf('Num Errors = %.0f',num_wrong))
    set(gcf,'color','white')
    grid on
```

## SET EIGEN INPUTS

```
function [ validation_input, train_input, test_input,validation_target, train_target, test_target,e_weights,
e_targets, normalization_factors ] = set_eigen_inputs( e_weights )
%SET_EIGEN_INPUTS: Take in the eigen weights for all the images
% outputs: training validation and test inputs and targets
% normalization_factors = [min mean max]

% initialize output arguments
   [validation_input, train_input, test_input] = deal([]);
   [validation_target, train_target, test_target] = deal([]);

% remove tag from input
   tag(1,:) = e_weights(1,:);
   e_targets = tag;
   e_weights = e_weights(2:end,:);

% normalize inputs
   [vectors, eyes] = size(e_weights);
   maximum = max(max(e_weights));
   minimum = min(min(e_weights));
   average = sum(sum(e_weights))./(vectors*eyes);

% create validation test and train inputs
   for i = 1:max(tag)-1
      index = find(tag == i,1, 'last');
      validation_input = [validation_input, [tag(index);  e_weights(:,index)]];
      test_input = [test_input, [tag(index-1);  e_weights(:,index-1)]];
      index = find(tag == i);
      index = index(1:end-2);
      train_input = [train_input, [tag(index);  e_weights(:,index)]];
   end

% convert the decimal targets to binary
   validation_tar = dec2bin(validation_input(1,:))';
   for i = 1:length(validation_tar)
      validation_target = [validation_target, bin2dec(validation_tar(:,i))];
   end

   test_tar = dec2bin(test_input(1,:))';
   for i = 1:length(validation_tar)
      test_target = [test_target, bin2dec(test_tar(:,i))];
   end

   train_tar = dec2bin(train_input(1,:))';
   for i = 1:length(train_tar)
      train_target = [train_target, bin2dec(train_tar(:,i))];
   end

% remove tag and set output arguments
   normalization_factors = [minimum, average, maximum];
   train_input =  train_input(2:end,:);
   test_input = test_input(2:end,:);
```

```
    validation_input = validation_input(2:end,:);
end
```

## TEST NEURAL NETWORK

```
clear all; close all; clc;
```

Load saved weights obtained from training

```
load tempweightsDCT2.mat
weights = net_weights;
```

Declare neural network parameters

```
number_of_eyes = 5;
BIAS = 1;
ALPHA = .8;
num_coef = 90;
ROTATE = 0; % 0or 1 for rotation
```

Run the test and validation images through the neural network

```
for test_type = 1:2;
```

```
if test_type ==2
    type_of_test = 'val';
    type = 'validation';
else
    type_of_test = 'test';
    type = 'test';
end
```

Choose rotated images or non-rotated test images

```
if ROTATE == 1
    filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals Week Combination\','WVT_coefficients_rotated.mat'];
else
    filename=['C:\Users\Kevin Haskett\Documents\College\Graduate School\Thesis Iris Biometric\Finals Week Combination\','WVT_coefficients.mat'];
end
load(filename)
```

Separate images into test validation and train

```
[ validation_input, train_input, test_input,...
    validation_target, train_target, test_target,e_weights,e_targets, normalization_factors ]...
    = set_eigen_inputs( WVT_weights );
```

Select number of eyes for test based on inputs from user

```
num_eyes = number_of_eyes*5-1;
num_diff_eyes = floor((num_eyes+1)/5) ;
```

Select target based on test or validation input from user

```
if strcmp('test',type)
   target = test_target(5:7,1:num_diff_eyes);
   input = test_input(1:num_coef,1:num_diff_eyes);
else
   target = validation_target(5:7,1:num_diff_eyes);
   input = validation_input(1:num_coef,1:num_diff_eyes);
end
```

Fun inputs through neural network and obtain output

```
[output] = net_simulation(weights, input,target,BIAS,ALPHA);
```

Display theoretical and experimental outputs

```
if test_type ==1
   target;
end

WRONG(test_type) = sum(sum(target-(output>.7)));
output>.7
```

---

```
end
```

NEURAL NETWORK SIMULATION

```
function [output] = net_simulation(weights, inputs,targets, BIAS,ALPHA,BETA)
```

```
 PHI = @(v) 1./(1+exp(-ALPHA.*v)); % log sig
%PHI = @(v) ALPHA.*((exp(BETA.*v)-exp(-BETA.*v))./((exp(BETA.*v)+exp(-BETA.*v)))); % tansig
```

Initialize the Neural Network with parameters

```
    validation_tar = targets;
    net_weights = weights;
    validation_in = inputs;
    [~,N] = size(weights);
    layers = zeros(1,N+1);
    layers(1) = length(inputs(:,1));
```

Run inputs through forward propagation part of Neural Network

```
for i = 2:N+1
    layers(i) = sum(weights(:,i-1)~=0)/(layers(i-1)+1);
end
    [~,cases] = size(validation_in);
    [num_tar,~] = size(validation_tar);
    net_output = zeros(num_tar,cases);

for test = 1:cases
    val_x = zeros(max(layers)+1,length(layers));
    val_x(1:length(validation_in(:,test))+1,1)= [BIAS; validation_in(:,test)];
    [val_v, ~] = deal(zeros(max(layers),length(layers)));
    val_v(1:length(validation_in(:,test)))=validation_in(:,test);

    len = length(validation_in(:,test))+1;
    for LAYER = 2:length(layers)
        num_w = layers(LAYER-1)+1;
        for NEURON = 1:layers(LAYER);
            val_v(NEURON,LAYER) = val_x(1:len,LAYER-1)'*...
                net_weights(NEURON*num_w-(num_w-1):NEURON*num_w,LAYER-1);
        end
        len = layers(LAYER)+1;
        val_x(1:len,LAYER)=[BIAS; PHI(val_v(1:len-1,LAYER))];
    end
    net_output(:,test) = val_x(2:1+num_tar,end);
end
```

return the output from Neural Network

```
output = net_output;
```

# APPENDIX B: DETAILED CHART DATA

**TABLE 32: Neural Network Size Analysis**

| | Layer 1 | Layer 2 | Layer 3 | Connections | Time (sec) |
|---|---|---|---|---|---|
| **1** | 5 | 5 | | 640 | 23274 |
| **2** | 5 | 5 | 5 | 670 | 11402 |
| **3** | 10 | 10 | | 1330 | 4025 |
| **4** | 10 | 10 | 10 | 1440 | 2912 |
| **5** | 15 | 15 | | 2070 | 1730 |
| **6** | 15 | 15 | 15 | 2310 | 1716 |
| **7** | 20 | 20 | | 2860 | 2602 |
| **8** | 15 | 21 | 11 | 3068 | 863 |
| **9** | 20 | 20 | 20 | 3280 | 1383 |
| **10** | 25 | 25 | | 3700 | 2727 |
| **11** | 22 | 25 | 25 | 3909 | 1914 |
| **12** | 25 | 25 | 25 | 4350 | 1231 |
| **13** | 30 | 30 | | 4590 | 1792 |
| **14** | 30 | 30 | 30 | 5520 | 1496 |
| **15** | 35 | 35 | | 5530 | 5236 |
| **16** | 40 | 40 | | 6520 | 12656 |
| **17** | 35 | 35 | 35 | 6790 | 17877 |

**TABLE 33: Fourier Input Analysis**

| Coefficients | Convergence (sec) |
|---|---|
| 49 | 597.64 |
| 81 | 338 |
| 121 | 49.41 |
| 169 | 55.89 |
| 225 | 72.64 |
| 289 | 100.48 |
| 361 | 157.83 |

**TABLE 34: α Optimization**

| ETA | time (min) |
|---|---|
| 0.1 | 5.28 |
| 0.2 | 6.51 |
| 0.3 | 3.86 |
| 0.4 | 2.91 |
| 0.5 | 1.60 |
| 0.6 | 1.10 |
| 0.7 | 1.60 |
| 0.8 | 1.74 |
| 0.9 | 2.13 |

**TABLE 35: η Optimization**

| alpha | time (sec) |
|---|---|
| 0.4 | 330 |
| 0.5 | 103 |
| 0.6 | 56.3 |
| 0.7 | 39.2 |
| 0.8 | 26 |
| 0.9 | 31 |
| 1 | 45 |

**TABLE 36: Summary Information on Rotational Distortion Analysis**

| Rotation | FT | Eigen | DCT | DWT | FT+E | FT+DWT | E+DCT | DCT+DWT |
|---|---|---|---|---|---|---|---|---|
| -16 | 0.6 | 0.7 | 0.8 | 0.7 | 0.4 | 0.7 | 0.7 | 0.6 |
| -14 | 0.5 | 0.8 | 0.7 | 0.7 | 0.4 | 0.7 | 0.8 | 0.7 |
| -12 | 0.4 | 0.8 | 0.7 | 0.7 | 0.4 | 0.7 | 0.8 | 0.9 |
| -10 | 0.4 | 0.8 | 0.7 | 0.7 | 0.4 | 0.7 | 0.8 | 1 |
| -8 | 0.4 | 0.8 | 0.6 | 0.8 | 0.4 | 0.7 | 0.7 | 0.9 |
| -6 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.8 | 0.7 | 0.9 |
| -4 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.8 | 0.9 |
| -2 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.8 | 0.9 |
| 0 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.7 | 0.9 | 1 |
| 2 | 0.3 | 0.8 | 0.7 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 4 | 0.3 | 0.8 | 0.6 | 0.9 | 0.4 | 0.6 | 0.8 | 1 |
| 6 | 0.4 | 0.7 | 0.5 | 0.9 | 0.4 | 0.7 | 0.8 | 1 |
| 8 | 0.4 | 0.7 | 0.5 | 0.8 | 0.4 | 0.8 | 0.8 | 1 |
| 10 | 0.4 | 0.7 | 0.6 | 0.6 | 0.4 | 0.7 | 0.6 | 1 |
| 12 | 0.3 | 0.7 | 0.8 | 0.6 | 0.4 | 0.6 | 0.6 | 0.9 |
| 14 | 0.3 | 0.7 | 0.7 | 0.6 | 0.4 | 0.6 | 0.6 | 0.9 |
| 16 | 0.4 | 0.7 | 0.6 | 0.6 | 0.4 | 0.6 | 0.6 | 0.9 |

**TABLE 37: Summary Information on Gaussian Blur Distortion Analysis**

| SNR | FT | Eigen | DCT | DWT | FT+E | FT+DWT | E+DCT | DCT+DWT |
|---|---|---|---|---|---|---|---|---|
| 30 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 28 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 26 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 24 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 22 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 20 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 18 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 16 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 14 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 12 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 10 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 9 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 0.9 |
| 8 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 0.9 |
| 7 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 0.9 |
| 6 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 0.9 |
| 5 | 0.5 | 0.7 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 0.9 |

**TABLE 38: Summary Information on Salt and Pepper Noise Analysis**

| SNR | FT | Eigen | DCT | DWT | FT+E | FT+DWT | E+DCT | DCT+DWT |
|---|---|---|---|---|---|---|---|---|
| 30 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 28 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 26 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 24 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 22 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 1 |
| 20 | 0.5 | 0.8 | 0.9 | 0.9 | 0.4 | 0.6 | 0.9 | 0.9 |
| 18 | 0.5 | 0.8 | 0.9 | 0.9 | 0.5 | 0.6 | 0.9 | 1 |
| 16 | 0.5 | 0.6 | 0.9 | 0.9 | 0.4 | 0.6 | 0.8 | 0.9 |
| 14 | 0.5 | 0.5 | 0.9 | 0.9 | 0.3 | 0.6 | 0.8 | 0.9 |
| 12 | 0.5 | 0.5 | 0.9 | 0.9 | 0.2 | 0.6 | 0.7 | 0.9 |
| 10 | 0.5 | 0.4 | 0.9 | 0.9 | 0.1 | 0.6 | 0.7 | 0.9 |
| 9 | 0.5 | 0.3 | 0.9 | 0.9 | 0.1 | 0.6 | 0.7 | 0.9 |
| 8 | 0.5 | 0.2 | 0.9 | 0.9 | 0.1 | 0.6 | 0.7 | 0.7 |
| 7 | 0.5 | 0.2 | 0.9 | 0.6 | 0.1 | 0.6 | 0.7 | 0.4 |
| 6 | 0.5 | 0.2 | 0.9 | 0.4 | 0.1 | 0.6 | 0.6 | 0.4 |
| 5 | 0.5 | 0.2 | 0.9 | 0.2 | 0.1 | 0.6 | 0.6 | 0.4 |